# IMAQ™

# IMAQ Vision for Measurement Studio™ User Manual

**LabWindows/CVI**

NATIONAL INSTRUMENTS™

May 2001 Edition
Part Number 323022A-01

**Worldwide Technical Support and Product Information**

ni.com

**National Instruments Corporate Headquarters**

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

**Worldwide Offices**

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

# Important Information

## Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

## Copyright



## Trademarks

CVI™, IMAQ™, LabVIEW™, Measurement Studio™, National Instruments™, ni.com™, and NI-IMAQ™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

## WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

# Conventions

The following conventions are used in this manual:

| | |
|---|---|
| » | The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box. |
|  | This icon denotes a note, which alerts you to important information. |
| **bold** | Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names. |
| *italic* | Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply. |
| `monospace` | Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts. |

# Contents

## Chapter 1
## Introduction to IMAQ Vision



## Chapter 2
## Getting Measurement-Ready Images



## Chapter 3
## Grayscale and Color Measurements

# 1

# Introduction to IMAQ Vision

This chapter describes the IMAQ Vision for LabWindows/CVI software and associated software products, discusses the documentation and examples available, outlines the IMAQ Vision function organization, and lists the steps for making a machine vision application.

**Note** For information about the system requirements and installation procedure for IMAQ Vision for LabWindows/CVI, see the *IMAQ Vision for Measurement Studio Release Notes* that came with your software.

## About IMAQ Vision

IMAQ Vision for Measurement Studio is a National Instruments product comprised of IMAQ Vision for LabWindows/CVI and IMAQ Vision for Visual Basic. IMAQ Vision for LabWindows/CVI is a library of C functions that you can use to develop machine vision and scientific imaging applications. IMAQ Vision for Visual Basic is a collection of ActiveX controls that offer the same imaging functionality as IMAQ Vision for LabWindows/CVI. National Instruments also offers IMAQ Vision for LabVIEW, which is a library of LabVIEW VIs for developing machine vision and scientific imaging applications. IMAQ Vision Builder, another software product from National Instruments, allows you to prototype your application strategy quickly without having to do any programming.

## Documentation and Examples

In addition to this manual, several documentation resources are available to help you create your vision application:

- *IMAQ Vision Concepts Manual*—If you are new to machine vision and imaging, read this manual to understand the concepts behind IMAQ Vision.
- IMAQ Vision for LabWindows/CVI function reference—If you need information about IMAQ Vision functions while creating your application, refer to this help file. Access this file from the **Start** menu by selecting **Programs»National Instruments»Vision»**

**Documentation»IMAQ Vision for LabWindows/CVI Function Reference**.

- Example programs—If you want examples of how to create specific applications, go to `cvi\samples\vision`.
- Application Notes—If you want to know more about advanced IMAQ Vision concepts and applications, refer to the Application Notes located on the National Instruments Web site at `ni.com/appnotes.nsf/`
- NI Developer Zone (NIDZ)—If you want even more information about developing your vision application, visit the NI Developer Zone at `ni.com/zone`. The NI Developer Zone contains example programs, tutorials, technical presentations, the Instrument Driver Network, a measurement glossary, an online magazine, a product advisor, and a community area where you can share ideas, questions, and source code with vision developers around the world.

# Application Development Environments

This release of IMAQ Vision for LabWindows/CVI supports the following Application Development Environments (ADEs) for Windows 2000/NT/Me/9*x*.

- LabWindows/CVI version 5.0.1 and higher
- Borland C++ Builder 3.0 and higher
- Microsoft Visual C/C++ version 6.0 and higher

**Note** Although IMAQ Vision has been tested and found to work with these ADEs, other ADEs may also work.

# IMAQ Vision Function Tree

The IMAQ Vision function tree (`NIVision.lfp`) contains separate classes corresponding to groups or types of functions. Table 1-1 lists the IMAQ Vision function types and gives a description of each type.

**Table 1-1.** IMAQ Vision Function Types

| Function Type | Description |
|---|---|
| Image Management | Functions that create space in memory for images and perform basic image manipulation. |
| Memory Management | A function that returns memory you no longer need to the operating system. |
| Error Management | Functions that set the current error, return the name of the function in which the last error occurred, return the error code of the last error, and clear any pending errors. |
| Acquisition | Functions that acquire images through an IMAQ hardware device. |
| Display | Functions that cover all aspects of image visualization and image window management. |
| Overlay | Functions that create and manipulate overlays. |
| Regions of Interest | Functions that create and manipulate regions of interest. |
| File I/O | Functions that read and write images to and from files. |
| Calibration | Functions that learn calibration information and correct distorted images. |
| Image Analysis | Functions that compute the centroid of an image, profile of a line of pixels, and the mean line profile. This type also includes functions that calculate the pixel distribution and statistical parameters of an image. |
| Grayscale Processing | Functions for grayscale image processing and analysis. |
| Binary Processing | Functions for binary image processing and analysis. |
| Color Processing | Functions for color image processing and analysis. |
| Pattern Matching | Functions that learn patterns and search for patterns in images. |
| Caliper | Functions designed for gauging, measurement, and inspection applications. |
| Operators | Functions that perform arithmetic, logic, and comparison operations with two images or with an image and a constant value. |
| Analytic Geometry | Functions that perform basic geometric calculations on an image. |

**Table 1-1.** IMAQ Vision Function Types (Continued)

| Function Type | Description |
|---|---|
| Frequency Domain Analysis | Functions for the extraction and manipulation of complex planes. Functions of this type perform FFTs, inverse FFTs, truncation, attenuation, addition, subtraction, multiplication, and division of complex images. |
| Barcode | A function that reads a barcode. |
| LCD | Functions that find and read seven-segment LCD characters. |
| Meter | Functions that return the arc information of a meter and read the meter. |
| Utilities | Functions that return structures, and a function that returns a pointer to predefined convolution matrices. |
| Obsolete | Functions that are no longer necessary but may exist in older applications. |

# IMAQ Machine Vision Function Tree

The IMAQ Machine Vision function tree (`NIMachineVision.fp`) contains separate classes corresponding to groups or types of functions. Table 1-2 lists the IMAQ Machine Vision function types and gives a description of each type.

**Table 1-2.** IMAQ Machine Vision Function Types

| Function Type | Description |
|---|---|
| Coordinate Transform | Functions that find coordinate transforms based on image contents. |
| Count and Measure Objects | A function that counts and measures objects in an image. |
| Find Patterns | A function that finds patterns in an image. |
| Locate Edges | Functions that locate different types of edges in an image. |
| Measure Distances | Functions that measure distances between objects in an image. |
| Measure Intensities | Functions that measure light intensities in various shaped regions within an image. |
| Select Region of Interest | Functions that allow a user to select a specific region of interest in an image. |

# Creating IMAQ Vision Applications

Figures 1-1 and 1-2 illustrate the steps for creating an application with IMAQ Vision. Figure 1-1 describes the general steps to designing a Vision application. The last step in Figure 1-1 is expanded upon in Figure 1-2. You can use a combination of the items in the last step to create your IMAQ Vision application. For more information about items in either diagram, see the corresponding chapter listed to the right of the item.

**Note** Diagram items enclosed with dashed lines are optional steps.

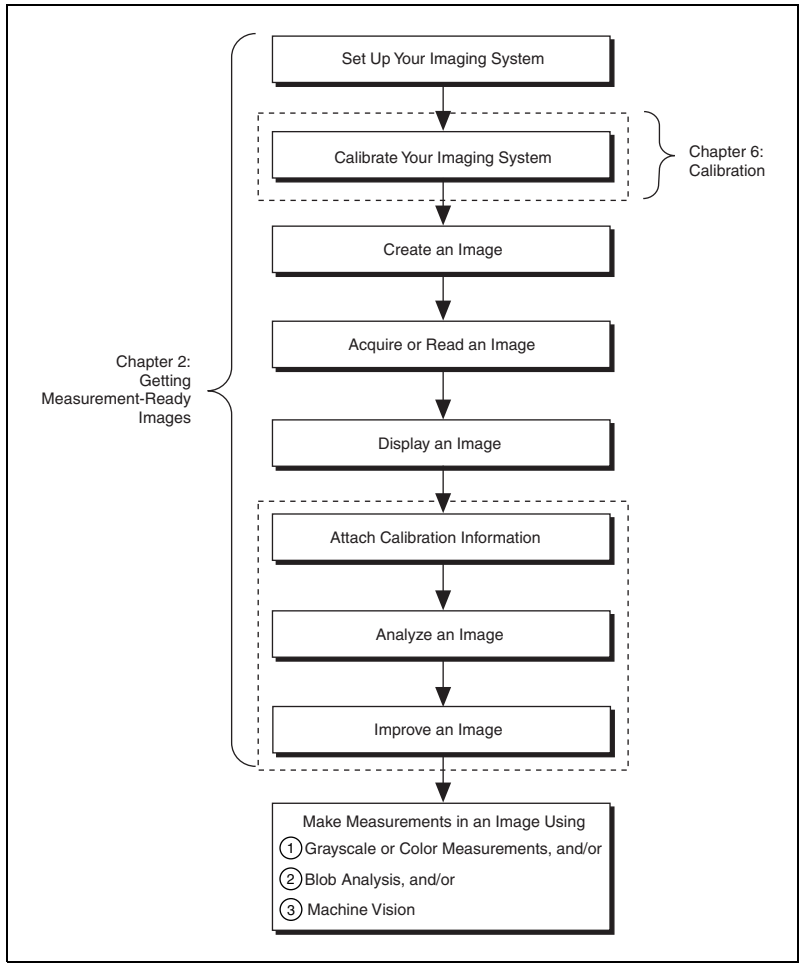



**Figure 1-1.** General Steps to Designing a Vision Application

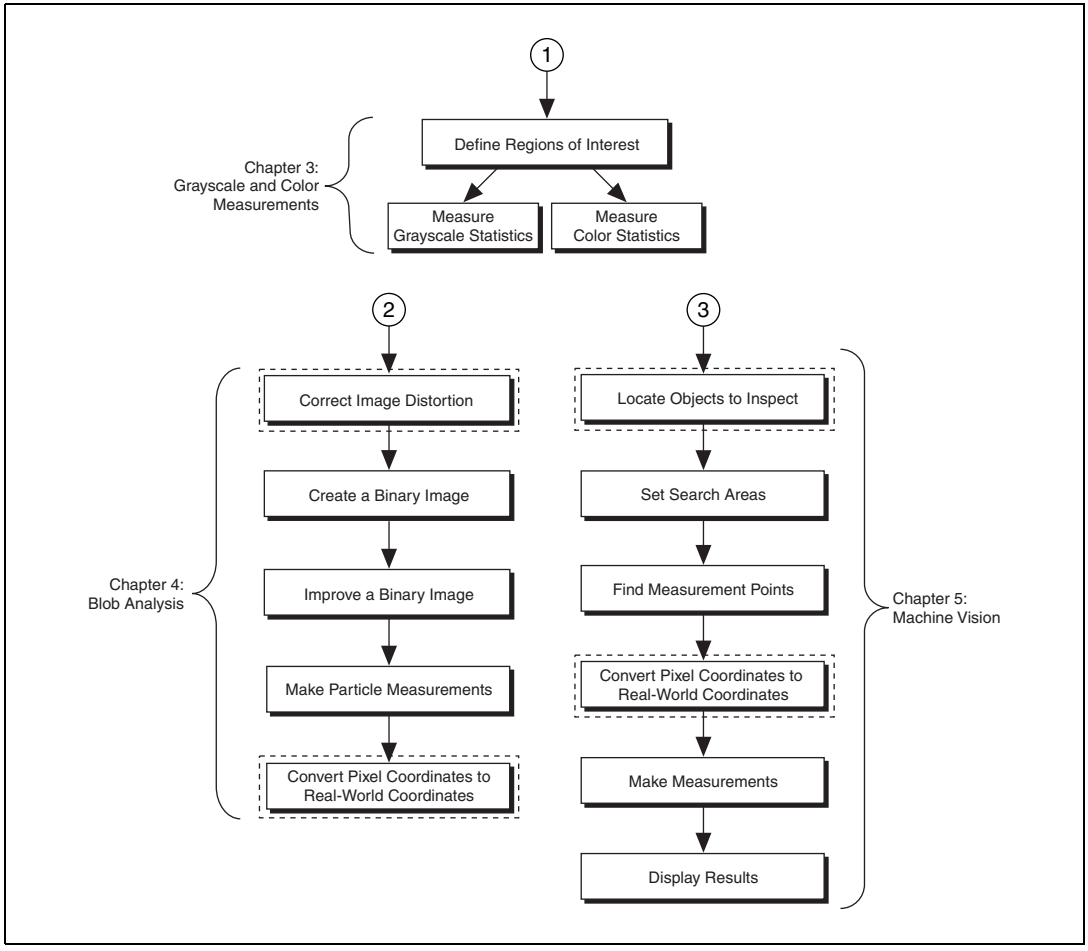



**Figure 1-2.** Inspection Steps for Building a Vision Application

# 2

# Getting Measurement-Ready Images

This chapter describes how to set up your imaging system, acquire and display an image, analyze the image, and prepare the image for additional processing.

## Set Up Your Imaging System

Before you acquire, analyze, and process images, you must set up your imaging system. The manner in which you set up your system depends on your imaging environment and the type of analysis and processing you need to do. Your imaging system should produce images with high enough quality so that you can extract the information you need from the images.

Follow the guidelines below to setup your imaging system.

1. Determine the type of equipment you need given your space constraints and the size of the object you need to inspect. For more information, see Chapter 3, *System Setup and Calibration*, of the *IMAQ Vision Concepts Manual*.
    a. Make sure your camera sensor is large enough to satisfy your minimum resolution requirement.
    b. Make sure your lens has a depth of field high enough to keep all of your objects in focus regardless of their distance from the lens. Also, make sure your lens has a focal length that meets your needs.
    c. Make sure your lighting provides enough contrast between the object under inspection and the background for you to extract the information you need from the image.
2. Position your camera so that it is parallel to the object under inspection. If your camera acquires images of the object from an angle, perspective errors occur. You can compensate for these errors with software, but using a parallel inspection angle obtains the fastest and most accurate results.

3. Select an image acquisition device that meets your needs. National Instruments offers several image acquisition (IMAQ) devices, such as analog color and monochrome IMAQ devices as well as digital devices. Visit `ni.com/imaq` for more information about IMAQ devices.

4. Configure the driver software for your image acquisition device. If you have a National Instruments image acquisition device, configure your NI-IMAQ driver software through Measurement & Automation Explorer (MAX). Open MAX by double-clicking the Measurement & Automation Explorer icon on your desktop. For more information, see the *NI-IMAQ User Manual* and the MAX online help.

# Calibrate Your Imaging System

After you set up your imaging system, you may want to calibrate your system. Calibrate your imaging system to assign real-world coordinates to pixel coordinates and compensate for perspective and nonlinear errors inherent in your imaging system.

Perspective errors occur when your camera axis is not perpendicular to the object under inspection. Nonlinear distortion may occur from aberrations in the camera lens. Perspective errors and lens aberrations cause images to appear distorted. This distortion misplaces information in an image, but it does not necessarily destroy the information in the image.

Use simple calibration if you only want to assign real-world coordinates to pixel coordinates. Use perspective and nonlinear distortion calibration if you need to compensate for perspective errors and nonlinear lens distortion. For detailed information about calibration, see Chapter 6, *Calibration*.

# Create an Image

To create an image in IMAQ Vision for LabWindows/CVI, call `imaqCreateImage()`. This function returns an image reference you can use when calling other IMAQ Vision functions. The only limitation to the size and number of images you can acquire and process is the amount of memory on your computer. When you create an image, specify the type of the image. Table 2-1 lists the valid image types.

**Table 2-1.** IMAQ Vision for LabWindows/CVI Image Types

| Value | Description |
|---|---|
| IMAQ_IMAGE_U8 | 8 bits per pixel—unsigned, standard monochrome |
| IMAQ_IMAGE_I16 | 16 bits per pixel—signed, monochrome |
| IMAQ_IMAGE_SGL | 32 bits per pixel—floating point, monochrome |
| IMAQ_IMAGE_COMPLEX | 2 × 32 bits per pixel—floating point, native format after a Fast Fourier Transform (FFT) |
| IMAQ_IMAGE_RGB | 32 bits per pixel—standard color |
| IMAQ_IMAGE_HSL | 32 bits per pixel—color |

If you plan to use filtering or blob analysis functions on the image, set the appropriate border size for the image. The default border size is 3.

When you create an image, IMAQ Vision creates an internal image structure to hold properties of the image, such as its name and border size. However, no memory is allocated to store the image pixels at this time. IMAQ Vision functions automatically allocate the appropriate amount of memory when the image size is modified. For example, functions that acquire or resample an image alter the image size, so they allocate the appropriate memory space for the image pixels. The return value of `imaqCreateImage()` is a pointer to the image structure. Supply this pointer as an input to all subsequent IMAQ Vision functions.

Most functions belonging to the IMAQ Vision library require one or more image pointers. The number of image pointers a function takes depends on the image processing function and the type of image you want to use. Some IMAQ Vision functions act directly on the image and require only one image pointer. Other functions that process the contents of images require pointers to the source image(s) and to a destination image.

You can create multiple images by executing `imaqCreateImage()` as many times as you want. Determine the number of required images through an analysis of your intended application. The decision is based on different processing phases and your need to keep the original image (after each processing step).

At the end of your application, dispose of each image that you created using `imaqDispose()`.

## Source and Destination Images

Some IMAQ Vision functions that modify the contents of an image have source image and destination image input parameters. The source image receives the image to process. The destination image receives the processing results. The destination image can receive either another image or the original, depending on your goals. If you do not want the contents of the original image to change, use separate source and destination images. If you want to replace the original image with the processed image, pass the same image as both the source and destination.

Depending on the function, the image type of the destination image can be the same or different than the image type of the source image. The function descriptions in the IMAQ Vision for LabWindows/CVI function reference help include the type of images you can use as image inputs and outputs. IMAQ Vision resizes the destination image to hold the result if the destination is not the appropriate size.

The following examples illustrate source and destination images with `imaqTranspose()`:

- `imaqTranspose(myImage, myImage);`

  This function creates a transposed image using the same image for the source and destination. The contents of `myImage` change.

- `imaqTranspose(myTransposedImage, myImage);`

  This function creates a transposed image and stores it in a destination different from the source. The `myImage` image remains unchanged, and `myTransposedImage` contains the result.

Functions that perform arithmetic or logical operations between two images have two source images and a destination image. You can perform an operation between two images and then either store the result in a separate destination image or in one of the two source images. In the latter case, make sure you no longer need the original data in the source image before storing the result over the data.

The following examples show the possible combinations using `imaqAdd()`:

- `imaqAdd(myResultImage, myImageA, myImageB);`

  This function adds two source images (`myImageA` and `myImageB`) and stores the result in a third image (`myResultImage`). Both source images remain intact after processing.

- `imaqAdd(myImageA, myImageA, myImageB);`

  This function adds two source images and stores the result in the first source image.

- `imaqAdd(myImageB, myImageA, myImageB);`

  This function adds two source images and stores the result in the second source image.

Most operations between two images require that the images have the same type and size. However, some arithmetic operations can work between two different types of images (for example, 8-bit and 16-bit).

Some functions perform operations that populate an image. Examples of this type of operation include reading a file, acquiring an image from an IMAQ device, or transforming a 2D array into an image. This type of function can modify the size of an image.

Some functions take an additional **mask** parameter. The presence of a **mask** parameter indicates that the processing or analysis is dependent on the contents of another image (the image mask). The only pixels in the source image that are processed are those whose corresponding pixels in the image mask are non-zero. If an image mask pixel is 0, the corresponding source image pixel is not processed or analyzed. The image mask must be an 8-bit image.

If you want to apply a processing or analysis function to the entire image, pass NULL for the image mask. Passing the same image to both the source image and image mask also gives the same effect as passing NULL for the image mask, except in this case the source image must be an 8-bit image.

# Acquire or Read an Image

After you create an image reference, you can acquire an image into your imaging system in three ways. You can acquire an image with a camera through your image acquisition device, load an image from a file stored on your computer, or convert the data stored in a 2D array to an image. Functions that acquire images, load images from file, or convert data from a 2D array automatically allocate the memory space required to accommodate the image data.

## Acquiring an Image

Use one of the following methods to acquire images with a National Instruments image acquisition (IMAQ) device:

- Acquire a single image using `imaqEasyAcquire()`. When you call this function, it initializes the IMAQ device and acquires the next incoming video frame. Use this function for low-speed single capture applications where ease of programming is essential.
- Acquire a single image using `imaqSnap()`. When you call this function, it acquires the next incoming video frame on an IMAQ device you have already initialized using `imgInterfaceOpen()` and `imgSessionOpen()`. Use this function for high-speed single capture applications.
- Acquire images continually through a grab acquisition. Grab functions perform high-speed acquisitions that loop continually on one buffer. Use `imaqSetupGrab()` to start the acquisition. Use `imaqGrab()` to return a copy of the current image. Use `imaqStopAcquisition()` to stop the acquisition.
- Acquire a fixed number of images using a sequence acquisition. Set up the acquisition using `imaqSetupSequence()`. Use `imaqStartAcquisition()` to acquire the number of images you requested during setup. If you want to acquire only certain images, supply `imaqSetupSequence()` with a table describing the number of frames to skip after each acquired frame.
- Acquire images continually through a ringed buffer acquisition. Set up the acquisition using `imaqSetupRing()`. Use `imaqStartAcquisition()` to start acquiring images into the acquired ring buffer. To get an image from the ring, call `imaqExtractFromRing()` or `imaqCopyRing()`. Use `imaqStopAcquisition()` to stop the acquisition.

**Note** You must use the `imgClose()` to release resources associated with the image acquisition device.

## Reading a File

Use `imaqReadFile()` to open and read data from a file stored on your computer into the image reference. You can read from image files stored in several standard formats: BMP, TIFF, JPEG, PNG, and AIPD. The software automatically converts the pixels it reads into the type of image you pass in.

Use `imaqReadVisionFile()` to open an image file containing additional information, such as calibration information, template information for pattern matching, or overlay information. For more information about pattern matching templates and overlays, see Chapter 5, *Machine Vision*.

You can also use `imaqGetFileInfo()` to retrieve image properties—image size, recommended image type, and calibration units—without actually reading all the image data.

### Converting an Array to an Image

Use `imaqArrayToImage()` to convert a 2D array to an image. You can also use `imaqImageToArray()` to convert an image to a 2D array.

## Display an Image

Display an image in an external window using `imaqDisplayImage()`. You can display images in 16 different external windows. Use the other display functions to configure the appearance of each external window. Properties you can set include whether the window has scroll bars, is resizable, or has a title bar. You can also use `imaqMoveWindow()` to position the external image window at a particular location on you monitor. See the IMAQ Vision for LabWindows/CVI function reference for a complete list of display functions.

**Note** Image windows are not LabWindows/CVI panels. They are managed directly by IMAQ Vision.

You can use a color palette to display grayscale images by applying a color palette to the window. Use `imaqSetWindowPalette()` to set predefined color palettes. For example, if you need to display a binary image—an image containing particle regions with pixel values of 1 and a background region with pixel values of 0—apply the predefined binary palette. For more information about color palettes, see the Chapter 2, *Display*, of the *IMAQ Vision Concepts Manual*.

**Note** At the end of your application, you should close all open external windows using `imaqCloseWindow()`.

# Attach Calibration Information

If you want to attach the calibration information of the current setup to each image you acquire, use `imaqCopyCalibrationInfo()`. This function takes in a source image containing the calibration information and a destination image that you want to calibrate. The output image is your inspection image with the calibration information attached to it. For detailed information about calibration, see Chapter 6, *Calibration*.

**Note** Because calibration information is part of the image, it is propagated throughout the processing and analysis of the image. Functions that modify the image size (such as geometrical transforms) void the calibration information. Use `imaqWriteVisionFile()` to save the image and all of the attached calibration information to a file.

# Analyze an Image

Once you acquire and display an image, you may want to analyze the contents of the image for the following reasons:

- To determine whether the image quality is high enough for your inspection task.
- To obtain the values of parameters that you want to use in processing functions during the inspection process.

The histogram and line profile tools can help you analyze the quality of your images.

Use `imaqHistogram()` to analyze the overall grayscale distribution in the image. Use the histogram of the image to analyze two important criteria that define the quality of an image—saturation and contrast. If your image is underexposed (does not have enough light) the majority of your pixels will have low intensity values, which appear as a concentration of peaks on the left side of your histogram. If your image is overexposed (has too much light) the majority of your pixels will have a high intensity values, which appear as a concentration of peaks on the right side of your histogram. If your image has an appropriate amount of contrast, your histogram will have distinct regions of pixel concentrations. Use the histogram information to decide if the image quality is high enough to separate objects of interest from the background.

If the image quality meets your needs, use the histogram to determine the range of pixel values that correspond to objects in the image. You can use

this range in processing functions, such as determining a threshold range during blob analysis.

If the image quality does not meet your needs, try to improve the imaging conditions to get the desired image quality. You may need to re-evaluate and modify each component of your imaging setup: lighting equipment and setup, lens tuning, camera operation mode, and acquisition board parameters. If you reach the best possible conditions with your setup but the image quality still does not meet your needs, try to improve the image quality using the image processing techniques described in the *Improve an Image* section.

Use `imaqLineProfile()` to get the pixel distribution along a line in the image, or use `imaqROIProfile()` to get the pixel distribution along a one-dimensional path in the image. By looking at the pixel distribution, you can determine if the image quality is high enough to provide you with sharp edges at object boundaries. Also, you can determine if the image is noisy, and identify the characteristics of the noise.

If the image quality meets your needs, use the pixel distribution information to determine some parameters of the inspection functions you want to use. For example, use the information from the line profile to determine the strength of the edge at the boundary of an object. You can input this information into `imaqEdgeTool()` to find the edges of objects along the line.

# Improve an Image

Using the information you gathered from analyzing your image, you may want to improve the quality of your image for inspection. You can improve your image with lookup tables, filters, grayscale morphology, and Fast Fourier transforms.

## Lookup Tables

Apply *lookup table* (LUT) transformations to highlight image details in areas containing significant information at the expense of other areas. A LUT transformation converts input grayscale values in the source image into other grayscale values in the transformed image. IMAQ Vision provides four functions that directly or indirectly apply lookup tables to images:

- `imaqMathTransform()`—Converts the pixel values of an image by replacing them with values from a predefined lookup table. IMAQ Vision has seven predefined lookup tables based on mathematical

transformations. For more information about these lookup tables, see Chapter 5, *Image Processing*, in the *IMAQ Vision Concepts Manual*.

- `imaqLookup()`—Converts the pixel values of an image by replacing them with values from a user-defined lookup table.
- `imaqEqualize()`—Distributes the grayscale values evenly within a given grayscale range. Use IMAQ Equalize to increase the contrast in images containing few grayscale values.
- `imaqInverse()`—Inverts the pixel intensities of an image to compute the negative of the image. For example, use `imaqInverse()` before applying an automatic threshold to your image if the background pixels are brighter than the object pixels.

# Filters

Filter your image when you need to improve the sharpness of transitions in the image or increase the overall signal-to-noise ratio of the image. You can choose either a lowpass or highpass filter depending on your needs.

Lowpass filters remove insignificant details by smoothing the image, removing sharp details, and smoothing the edges between the objects and the background. You can use `imaqLowpass()` or define your own lowpass filter with `imaqConvolve()` or `imaqNthOrderFilter()`.

Highpass filters emphasize details, such as edges, object boundaries, or cracks. These details represent sharp transitions in intensity value. You can define your own highpass filter with `imaqConvolve()` or `imaqNthOrderFilter()`, or you can use a predefined highpass filter with `imaqEdgeFilter()` or `imaqCannyEdgeFilter()`. The `imaqEdgeFilter()` function allows you to find edges in an image using predefined edge detection kernels, such as the Sobel, Prewitt, and Roberts kernels.

## Convolution Filter

The `imaqConvolve()` function allows you to use a predefined set of lowpass and highpass filters. Each filter is defined by a kernel of coefficients. Use `imaqGetKernel()` to retrieve predefined kernels. If the predefined kernels do not meet your needs, define your own custom filter using a 2D array of floating point numbers.

### Nth Order Filter

The `imaqNthOrderFilter()` function allows you to define a lowpass or highpass filter depending on the value of *N* that you choose. One specific Nth order filter, the median filter, removes speckle noise, which appears as small black and white dots. Use `imaqMedianFilter()` to apply a median filter. For more information about Nth order filters, see Chapter 5, *Image Processing*, of the *IMAQ Vision Concepts Manual*.

## Grayscale Morphology

Perform grayscale morphology when you want to filter grayscale features of an image. Grayscale morphology helps you remove or enhance isolated features, such as bright pixels on a dark background. Use these transformations on a grayscale image to enhance non-distinct features before thresholding the image in preparation for blob analysis.

Use `imaqGrayMorphology()` to perform one of the following seven transformations:

- Erosion—Reduces the brightness of pixels that are surrounded by neighbors with a lower intensity.
- Dilation—Increases the brightness of pixels surrounded by neighbors with a higher intensity. A dilation has the opposite effect as an erosion.
- Opening—Removes bright pixels isolated in dark regions and smooths boundaries.
- Closing—Removes dark pixels isolated in bright regions and smooths boundaries.
- Proper-opening—Removes bright pixels isolated in dark regions and smooths the inner contours of particles.
- Proper-closing—Removes dark pixels isolated in bright regions and smooths the inner contours of particles.
- Auto-median—Generates simpler particles that have fewer details.

For more information about grayscale morphology transformations, see Chapter 5, *Image Processing*, of the *IMAQ Vision Concepts Manual*.

## FFT

Use the Fast Fourier Transform (FFT) to convert an image into its frequency domain. In an image, details and sharp edges are associated with mid to high spatial frequencies because they introduce significant gray-level variations over short distances. Gradually varying patterns are associated with low spatial frequencies.

An image can have extraneous noise, such as periodic stripes, introduced during the digitization process. In the frequency domain, the periodic pattern is reduced to a limited set of high spatial frequencies. Also, the imaging setup may produce non-uniform lighting of the field of view, which produces an image with a light drift superimposed on the information you want to analyze. In the frequency domain, the light drift appears as a limited set of low frequencies around the average intensity of the image (DC component).

You can use algorithms working in the frequency domain to isolate and remove these unwanted frequencies from your image. Follow these steps to obtain an image in which the unwanted pattern has disappeared but the overall features remain:

1. Use `imaqFFT()` to convert an image from the spatial domain to the frequency domain. This function computes the Fast Fourier Transform (FFT) of the image and results in a complex image representing the frequency information of your image.
2. Improve your image in the frequency domain with a lowpass or highpass frequency filter. Specify which type of filter to use with `imaqAttenuate()` or `imaqTruncate()`. Lowpass filters smooth noise, details, textures, and sharp edges in an image. Highpass filters emphasize details, textures, and sharp edges in images, but they also emphasize noise.
    - Lowpass attenuation—The amount of attenuation is directly proportional to the frequency information. At low frequencies, there is little attenuation. As the frequencies increase, the attenuation increases. This operation preserves all of the zero frequency information. Zero frequency information corresponds to the DC component of the image or the average intensity of the image in the spatial domain.
    - Highpass attenuation—The amount of attenuation is inversely proportional to the frequency information. At high frequencies, there is little attenuation. As the frequencies decrease, the attenuation increases. The zero frequency component is removed entirely.
    - Lowpass truncation—Specify a frequency. The frequency components above the ideal cutoff frequency are removed, and the frequencies below it remain unaltered.
    - Highpass truncation—Specify a frequency. The frequency components above the ideal cutoff frequency remain unaltered, and the frequencies below it are removed.

3. To transform your image back to the spatial domain, use `imaqInverseFFT()`.

## Complex Image Operations

The `imaqExtractComplexPlane()` and `imaqReplaceComplexPlane()` functions allow you to access, process, and update independently the real and imaginary planes of a complex image. You can also convert planes of a complex image to an array and back with `imaqComplexPlaneToArray()` and `imaqArrayToComplexPlane()`.

# 3

# Grayscale and Color Measurements

This chapter describes how to take measurements from grayscale and color images. You can make inspection decisions based on image statistics, such as the mean intensity level in a region. Based on the image statistics, you can perform many machine vision inspection tasks on grayscale or color images, such as detecting the presence or absence of components, detecting flaws in parts, and comparing a color component with a reference. Figure 3-1 illustrates the basic steps involved in making grayscale and color measurements.

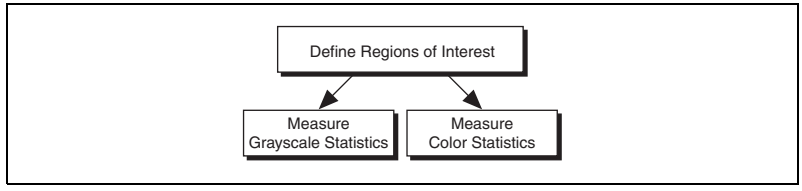



**Figure 3-1.** Steps to Taking Grayscale and Color Measurements

## Define Regions of Interest

A region of interest (ROI) is an area of an image in which you want to focus your image analysis. You can define an ROI interactively, programmatically, or with an image mask.

### Interactively Defining Regions

You can interactively define an ROI in a window that displays an image. Use the tools from the IMAQ Vision tools palette to interactively define and manipulate an ROI. Table 3-1 describes each of the tools and the manner in which you use them.

**Table 3-1.** Tools Palette Functions

| Icon | Tool Name | Function |
|---|---|---|
|  | Selection Tool | Select an ROI in the image and adjust the position of its control points and contours.<br>Action: Click on the desired ROI or control points. |
|  | Point | Select a pixel in the image.<br>Action: Click on the desired position. |
|  | Line | Draw a line in the image.<br>Action: Click on the initial position and click again on the final position. |
|  | Rectangle | Draw a rectangle or square in the image.<br>Action: Click on one corner and drag to the opposite corner. |
|  | Rotated Rectangle | Draw a rotated rectangle in the image.<br>Action: Click on one corner and drag to the opposite corner to create the rectangle. Then, click on the lines inside the rectangle and drag to adjust the rotation angle. |
|  | Oval | Draw an oval or circle in the image.<br>Action: Click on the center position and drag to the desired size. |
|  | Annulus | Draw an annulus in the image.<br>Action: Click on the center position and drag to the desired size. Adjust the inner and outer radii, and adjust the start and end angle. |
|  | Broken Line | Draw a broken line in the image.<br>Action: Click to place a new vertex and double-click to complete the ROI element. |
|  | Polygon | Draw a polygon in the image.<br>Action: Click to place a new vertex and double-click to complete the ROI element. |
|  | Freehand Line | Draw a freehand line in the image.<br>Action: Click on the initial position, drag to the desired shape and release the mouse button to complete the shape. |

**Table 3-1.** Tools Palette Functions (Continued)

| Icon | Tool Name | Function |
|---|---|---|
|  | Freehand | Draw a freehand region in the image.<br>Action: Click on the initial position, drag to the desired shape and release the mouse button to complete the shape. |
|  | Zoom | Zoom-in or zoom-out in an image.<br>Action: Click on the image to zoom in. Hold down <Shift> and click to zoom out. |
|  | Pan | Pan around an image.<br>Action: Click on an initial position, drag to the desired position and release the mouse button to complete the pan. |

Hold down <Shift> while drawing an ROI if you want to constrain the ROI to the horizontal, vertical, or diagonal axes when possible. Use the selection tool to position an ROI by its control points or vertices. ROIs are context sensitive, meaning that the cursor actions differ depending on the ROI with which you interact. For example, if you move your cursor over the side of a rectangle, the cursor changes to indicate that you can click and drag the side to resize the rectangle. If you want to draw more than one ROI in a window, hold down <Ctrl> while drawing additional ROIs.

You can display the IMAQ Vision tools palette as part of an ROI constructor window or in a separate, floating window. Follow these steps to invoke an ROI constructor and define an ROI from within the ROI constructor window:

1. Use `imaqConstructROI()` to display an image and the tools palette in an ROI constructor window, as shown in Figure 3-2.

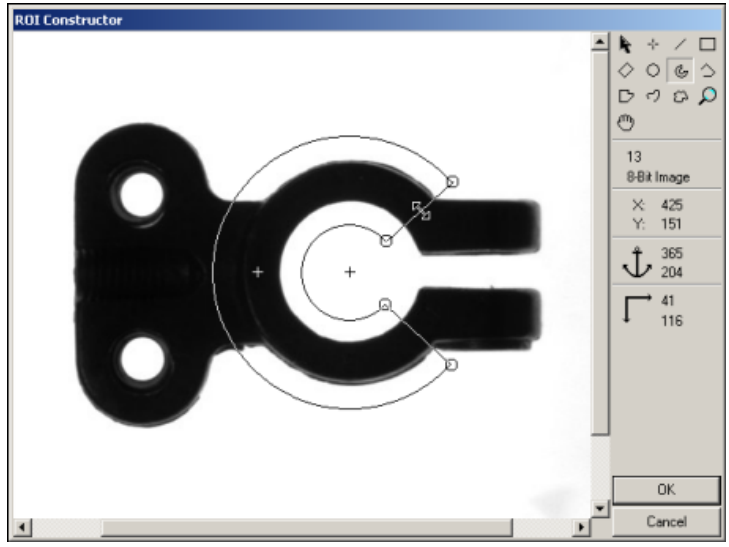



**Figure 3-2.** ROI Constructor

2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize and reposition the ROI until it designates the area you want to inspect.
4. Click the OK button to output a descriptor of the region you selected. You can input the ROI descriptor into many analysis and processing functions. You can also convert the ROI descriptor into an image mask, which you can use to process selected regions in the image. Use `imaqROIToMask()` to convert the ROI descriptor into an image mask.

The tools palette, shown in Figure 3-3, automatically transforms from the palette on the left to the palette on the right when you manipulate an ROI tool in an image window. The palette on the right displays the characteristics of the ROI you are drawing.

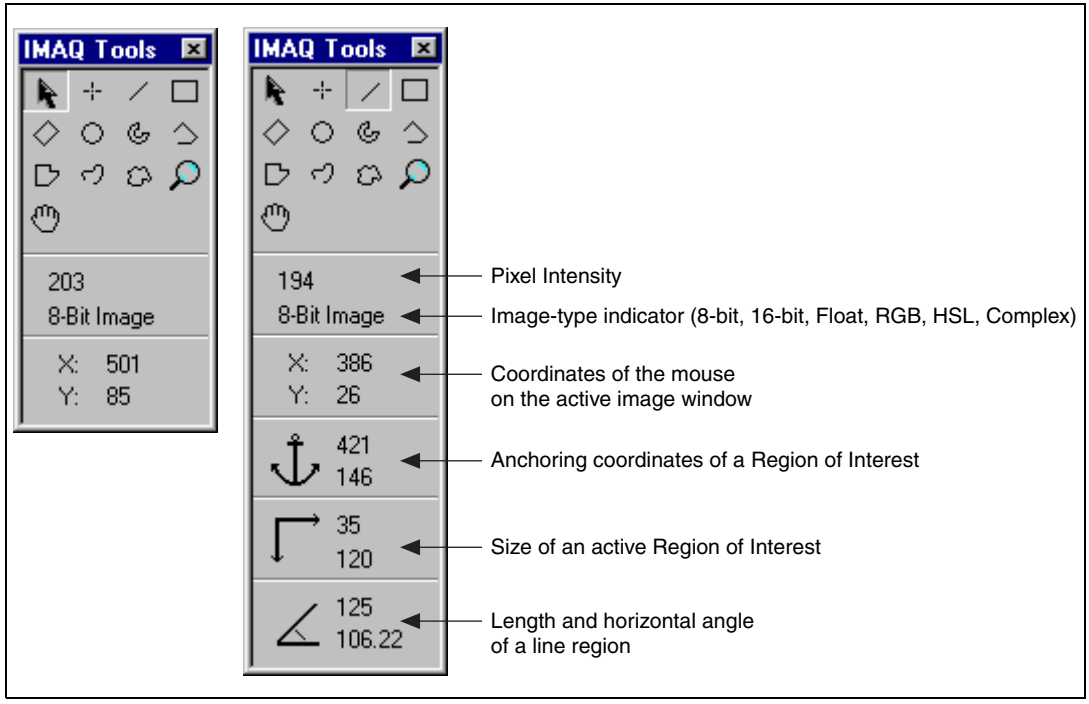



**Figure 3-3.** Tools Palette Tools and Information

The following list describes how you can display the tools palette in a separate window and manipulate the palette.

- Use `imaqShowToolWindow()` to display the tools window in a floating window.
- Use `imaqSetupToolWindow()` to configure the appearance of the tools window.
- Use `imaqMoveToolWindow()` to move the tools palette.
- Use `imaqCloseToolWindow()` to close the tools palette.

If you want to draw an ROI without using an ROI constructor or displaying the tools palette in a separate window, use `imaqSetCurrentTool()`. This function allows you to select a contour from the tools palette without opening the palette.

You can also use `imaqSelectPoint()`, `imaqSelectLine()`, `imaqSelectRect()`, and `imaqSelectAnnulus()` to define regions of interest. Follow these steps to use these functions:

1. Call the function to display an image in an ROI Constructor window. Only the tools specific to that function are available for you to use.
2. Draw an ROI on your image. Resize or reposition the ROI until it covers the area you want to process.
3. Click the OK button to populate a structure representing the ROI. You can use this structure as an input to a variety of functions, such as the following functions that measure grayscale intensity:
   - `imaqLightMeterPoint()`—Uses the output of `imaqSelectPoint()`
   - `imaqLightMeterLine()`—Uses the output of `imaqSelectLine()`
   - `imaqLightMeterRect()`—Uses the output of `imaqSelectRect()`

## Programmatically Defining Regions

When you have an automated application, you may need to define regions of interest programmatically. To programmatically define an ROI, create the ROI using `imaqCreateROI()` and then add the individual contours. A contour is a shape that defines an ROI. You can create contours from points, lines, rectangles, ovals, polygons, and annuli. For example, to add a rectangular contour to an ROI, use `imaqAddRectContour()`.

Specify regions by providing basic parameters that describe the region you want to define. For example, define a point by providing the x-coordinate and y-coordinate. Define a line by specifying the start and end coordinates. Define a rectangle by specifying the coordinates of the top, left point; the width and height; and the rotation angle (in the case of a rotated rectangle).

## Defining Regions with Masks

You can define regions to process with image masks. An image mask is an 8-bit image of the same size as or smaller than the image you want to process. Pixels in the mask image determine whether the corresponding pixel in the source image needs to be processed. If a pixel in the image mask has a value different than 0, the corresponding pixel in the source image is processed. If a pixel in the image mask has a value of 0, the corresponding pixel in the source image is left unchanged.

When you need to make intensity measurements on particles in an image, you can use a mask to define the particles. First, threshold your image to make a new binary image. For more information on binary images, see Chapter 4, *Blob Analysis*. You can input the binary image or a labeled version of the binary image as a mask image to the intensity measurement function. If you want to make color comparisons, convert the binary image into an ROI descriptor using `imaqMaskToROI()`.

# Measure Grayscale Statistics

You can measure grayscale statistics in images using light meters or quantitative analysis functions. You can obtain the center of energy for an image with the centroid function.

Use `imaqLightMeterPoint()` to measure the light intensity at a point in the image. Use `imaqLightMeterLine()` to get pixel value statistics along a line in the image, such as mean intensity, standard deviation, minimum intensity, and maximum intensity. Use `imaqLightMeterRect()` to get the pixel value statistics within a rectangular region in an image.

Use `imaqQuantify()` to obtain the following statistics about the entire image or individual regions in the image: mean intensity, standard deviation, minimum intensity, maximum intensity, area, and the percentage of the image that you analyzed. You can specify regions in the image with a labeled image mask. A labeled image mask is a binary image that has been processed so that each region in the image mask has a unique intensity value. Use `imaqLabel()` to label your image mask.

Use `imaqCentroid()` to compute the energy center of the image, or of a region within an image.

# Measure Color Statistics

Most image processing and analysis functions apply to 8-bit images. However, you can analyze and process individual components of a color image.

Using `imaqExtractColorPlanes()`, you can break down a color image into various sets of primary components, such as RGB (Red, Green, and Blue), HSI (Hue, Saturation, and Intensity), HSL (Hue, Saturation, and Luminance), or HSV (Hue, Saturation, and Value). Each component becomes an 8-bit image that you can process like any other grayscale

image. Using `imaqReplaceColorPlanes()`, you can reassemble a color image from a set of three 8-bit images, where each image becomes one of the three primary components. Figure 3-4 illustrates how a color image breaks down into its three components.

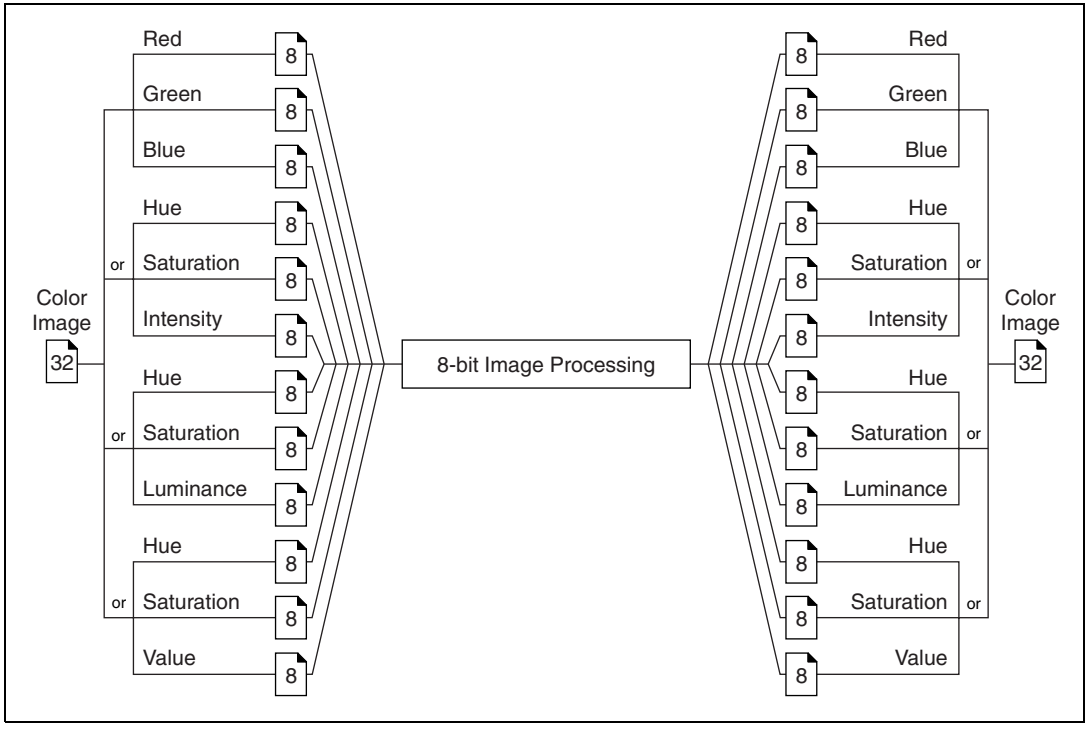



**Figure 3-4.** Primary Components of a Color Image

Use `imaqExtractColorPlanes()` to extract the red, green, blue, hue saturation, intensity, luminance, or value plane of a color image into an 8-bit image.

## Comparing Colors

You can use the color matching capability of IMAQ Vision to compare or evaluate the color content of an image or regions in an image.

Follow these steps to compare colors using color matching:

1. Select an image containing the color information that you want to use as a reference. The color information can consist of a single color or multiple dissimilar colors, such as red and blue.

2. Use the entire image or regions in the image to learn the color information using `imaqLearnColor()`, which outputs a color spectrum that contains a compact description of the color information that you learned. Use the color spectrum to represent the learned color information for all subsequent matching operations. See Chapter 14, *Color Inspection*, of the *IMAQ Vision Concepts Manual* for more information.
3. Define an entire image, a region, or multiple regions in an image as the inspection or comparison area.
4. Use `imaqMatchColor()` to compare the learned color information to the color information in the inspection regions. This function returns an array of scores that indicates how close the matches are to the learned color information.
5. Use the color matching score as a measure of similarity between the reference color information and the color information in the image regions being compared.

## Learning Color Information

When learning color information, you should choose the color information carefully:

- Specify an image or regions in an image that contain the color or color set that you want to learn.
- Select how detailed you want the color information to be learned.
- Choose colors that you want to ignore during matching.

### Choosing the Right Color Information

Because color matching only uses color information to measure similarity, the image or regions in the image representing the object should contain only the significant colors that represent the object, as shown in Figure 3-5a. Figure 3-5b illustrates an unacceptable region containing background colors.

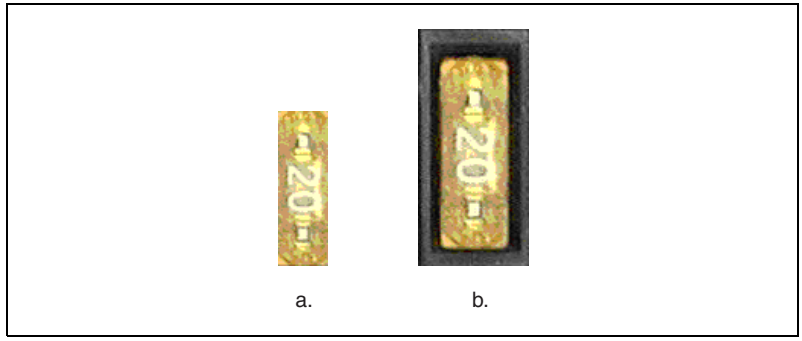



**Figure 3-5.** Template Color Information

## Specifying the Color Information to Learn

You can learn the color information associated with an entire image, a region in an image, or multiple regions in an image.

### Using the Entire Image

You can use an entire image to learn the color spectrum that represents the entire color distribution of the image. In a fabric identification application, for example, an entire image can specify the color information associated with a certain fabric type, as shown in Figure 3-6.

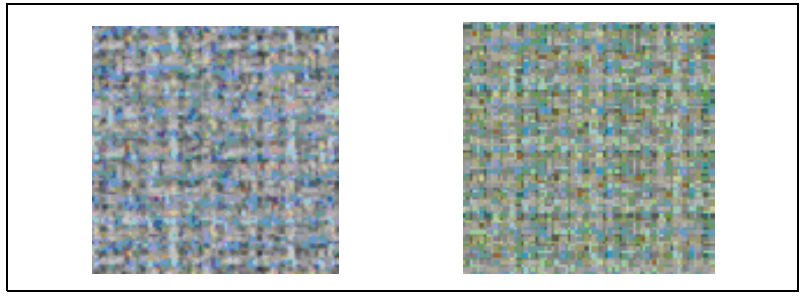


**Figure 3-6.** Using the Entire Image to Learn Color Distribution

### Using a Region in the Image

You can select a region in the image to provide the color information for comparison. A region is helpful for pulling out the useful color information in an image. Figure 3-7 shows an example of using a region that contains the color information that is important for the application.

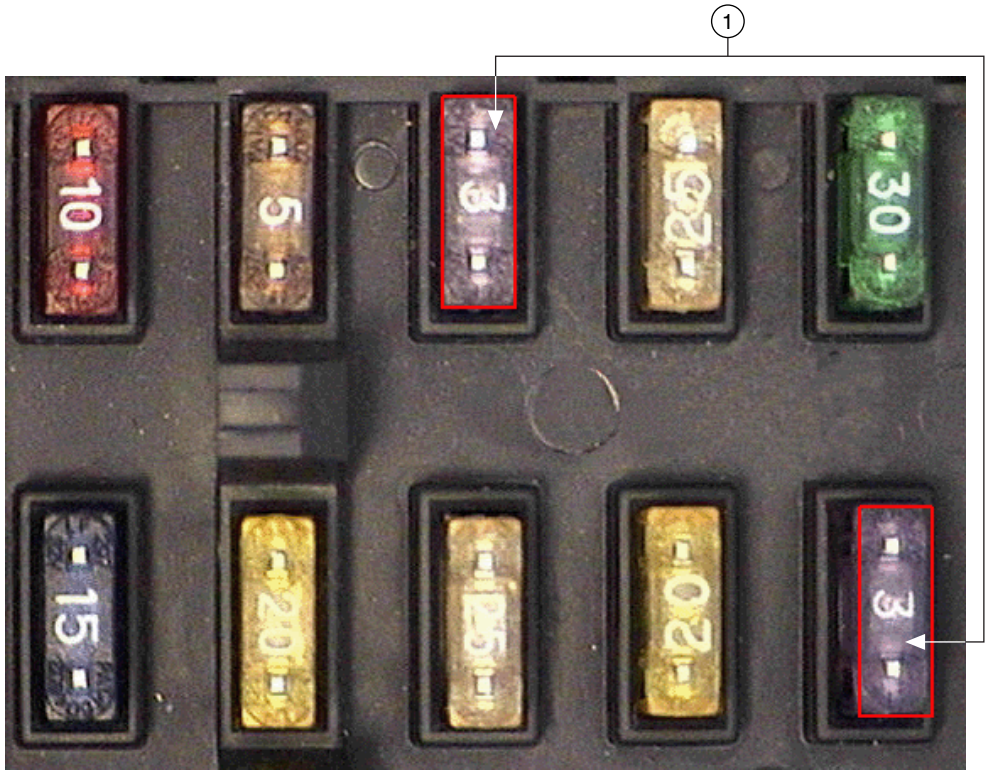
**Figure 3-7.** Using a Single Region to Learn Color Distribution

## Using Multiple Regions in the Image

The interaction of light with an object's surface creates the observed color of that object. The color of a surface depends on the directions of illumination and the direction from which the surface is observed. Two identical objects may have different appearances because of a difference in positioning or a change in the lighting conditions. Figure 3-8 shows how light reflects differently off of the 3D surfaces of the fuses, resulting in slightly different colors for identical fuses. (Compare the 3 amp fuse in the upper row with the 3 amp fuse in the lower row.) This results in different color spectra for identical fuses.

If you learn the color spectrum by drawing a region of interest around the 3 amp fuse in the upper row, and then do a color matching for the 3 amp fuse in the upper row, you get a very high match score for it (close to 1000). But the match score for the 3 amp fuse in the lower row is quite low (around 500). This problem could cause a mismatch for the color matching in a fuse box inspection process.

IMAQ Vision's color learning software uses a clustering process to find the representative colors from the color information specified by one or multiple regions in the image. To create a representative color spectrum for all 3 amp fuses in the learning phase, draw an ROI around the 3 amp fuse in the upper row, hold down <Shift>, and draw another ROI around the 3 amp fuse in the lower row. The new color spectrum results in similar, high match scores (around 800) for both 3 amp fuses. Use as many samples as you want in an image to learn the representative color spectrum for a specified template.




1 Regions used to learn color information

**Figure 3-8.** Using Multiple Regions to Learn Color Distribution

## Choosing a Color Representation Sensitivity

When you learn a color, you need to specify the sensitivity required to specify the color information. An image containing a few, well-separated colors in the color space requires a lower sensitivity to describe the color than an image that contains colors that are close to one another in the color space. Use the color sensitivity parameter of `imaqLearnColor()` to specify the granularity you want to use to represent the colors. For more information about color sensitivity, see the *Color Sensitivity* section of Chapter 5, *Machine Vision*.

## Ignoring Learned Colors

Ignore certain color components in color matching by replacing the corresponding component in the input color spectrum array to –1. For example, by replacing the last component in the color spectrum with –1, color matching ignores the color white. By replacing the second to last component in the color spectrum, color matching ignores the color black. To ignore other color components in color matching, determine the index to the color spectrum by locating the corresponding bins in the color wheel, where each bin corresponds to a component in the color spectrum array. Ignoring certain colors such as the background color results in a more accurate color matching score. Ignoring the background color also provides more flexibility when defining the regions of interest in the color matching process. Ignoring certain colors, such as white color created by glare on a metallic surface, also improves the accuracy of the color matching. Experiment learning the color information on different parts of the images to determine which colors to ignore. For more information about the color wheel and color bins, see Chapter 14, *Color Inspection*, in the *IMAQ Vision Concepts Manual*.

# 4

# Blob Analysis

This chapter describes how to perform blob (Binary Large Object) analysis on your images. Use blob analysis to find statistical information about blobs, such as the presence, size, number, and location of blob regions. With this information, you can perform many machine vision inspection tasks, such as detecting flaws on silicon wafers or detecting soldering defects on electronic boards. Examples of how blob analysis can help you perform web inspection tasks include locating structural defects on wood planks or detecting cracks on plastic sheets.

Figure 4-1 illustrates the steps involved in performing blob analysis. Diagram items enclosed with dashed lines are optional steps.

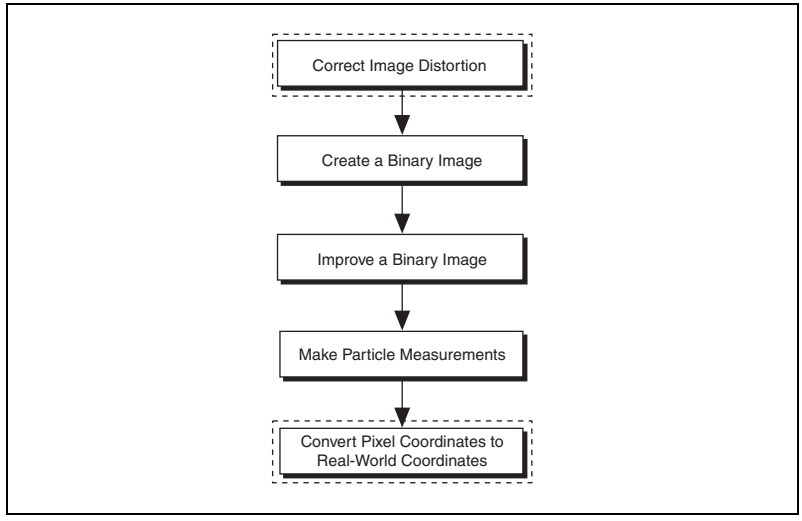



**Figure 4-1.** Steps to Performing Blob Analysis

# Correct Image Distortion

If you need to make accurate shape measurements based on the blobs in an image containing perspective and nonlinear distortion errors, correct the distortion using the calibration information you attached to your image. Use `imaqCorrectCalibratedImage()` to correct distortion in your grayscale image before thresholding it. See Chapter 6, *Calibration*, for more information about correcting an image using calibration information.

# Create a Binary Image

Threshold your grayscale or color image to create a binary image. Creating a binary image separates the objects that you want to inspect from the background. The threshold operation sets the background pixels to 0 in the binary image, while setting the object pixels to a non-zero value. Object pixels have a value of 1 by default, but you can set the object pixels to any value or retain their original value.

You can use different techniques to threshold your image. If all the objects of interest in your grayscale image fall within a continuous range of intensities and you can specify this threshold range manually, use `imaqThreshold()` to threshold your image.

If all the objects in your grayscale image are either brighter or darker than your background, you can use `imaqAutoThreshold()` to automatically determine the optimal threshold range and threshold your image. Automatic thresholding techniques offer more flexibility than simple thresholds based on fixed ranges. Because automatic thresholding techniques determine the threshold level according to the image histogram, the operation is more independent of changes in the overall brightness and contrast of the image than a fixed threshold. These techniques are more resistant to changes in lighting, which makes them well suited for automated inspection tasks.

If your grayscale image contains objects that have multiple discontinuous grayscale values, use `imaqMultithreshold()` to specify multiple threshold ranges.

If you need to threshold a color image, use `imaqColorThreshold()`. You must specify threshold ranges for each of the color planes (either Red, Green, and Blue or Hue, Saturation, and Luminance). The binary image resulting from a color threshold is an 8-bit binary image.

# Improve the Binary Image

After you threshold your image, you may want to improve the resulting binary image with binary morphology. You can use primary binary morphology or advanced binary morphology to remove unwanted blobs, separate connected blobs, or improve the shape of blobs. Primary morphology functions work on the image as a whole by processing pixels individually. Advanced morphology operations are built upon the primary morphological operators and work on particles as opposed to pixels.

**Note** The terms *blob* and *particle* are used interchangeably in this chapter.

The advanced morphology functions that improve binary images require that you specify the type of connectivity to use. Connectivity specifies how IMAQ Vision determines whether two adjacent pixels belong to the same particle. If you have a blob that contains narrow areas, use connectivity-8 to ensure that the software recognizes the connected pixels as one blob. If you have two blobs that touch at one point, use connectivity-4 to ensure that the software recognizes the pixels as two separate blobs. For more information about connectivity, see Chapter 9, *Binary Morphology*, of the *IMAQ Vision Concepts Manual*.

**Note** Use the same type of connectivity throughout your application.

## Removing Unwanted Blobs

Use `imaqRejectBorder()` to remove blobs that touch the border of the image. Reject blobs on the border of the image when you suspect that the information about those blobs is incomplete.

Use `imaqSizeFilter()` to remove large or small particles that do not interest you. You can also use the `IMAQ_ERODE`, `IMAQ_OPEN`, and `IMAQ_POPEN` methods in `imaqMorphology()` to remove small particles. Unlike `imaqSizeFilter()`, these three operations alter the size and shape of the remaining blobs.

Use the `IMAQ_HITMISS` method of `imaqMorphology()` to locate particular configurations of pixels, which you define with a structuring element. Depending on the configuration of the structuring element, the `IMAQ_HITMISS` method can locate single isolated pixels, cross-shape or longitudinal patterns, right angles along the edges of particles, and other user-specified shapes. For more information about structuring elements, see Chapter 9, *Binary Morphology*, of the *IMAQ Vision Concepts Manual*.

If you know enough about the shape features of the blobs you want to keep, use `imaqParticleFilter()` to filter out particles that do not interest you. If you do not have enough information about the particles you want to keep at this point in your processing, use the particle measurement functions to obtain this information before applying a particle filter. See the *Make Particle Measurements* section for more information about the measurement functions.

## Separating Touching Blobs

Use `imaqSeparation()` or apply an erosion or an open operation with `imaqMorphology()` to separate touching objects. The `imaqSeparation()` function is an advanced function that separates blobs without modifying their shapes. However, erosion and open functions alter the shape of all the blobs.

**Note** A separation is a time-intensive operation compared to an erosion or open operation. Consider using an erosion or open if speed is an issue with your application.

## Improving Blob Shapes

Use `imaqFillHoles()` to fill holes in the blobs. Use `imaqMorphology()` to perform a variety of operations on the blobs. You can use the `IMAQ_AUTOM`, `IMAQ_CLOSE`, `IMAQ_PCLOSE`, `IMAQ_OPEN`, and `IMAQ_POPEN` methods to smooth the boundaries of the blobs. Open and proper open smooth the boundaries of the blob by removing small isthmuses while close widens the isthmuses. Close and proper close fill small holes in the blob. Auto-median removes isthmuses and fills holes. For more information about these methods, see Chapter 9, *Binary Morphology*, in the *IMAQ Vision Concepts Manual*.

# Make Particle Measurements

After you create a binary image and improve it, you can make particle measurements. With these measurements you can determine the location of blobs and their shape features. Use the following functions to perform particle measurements:

- `imaqGetParticleInfo()`—This function returns the number of blobs in an image and a report containing the pixel area, real-world area, and bounding rectangle of the blobs. You can use the bounding rectangle to determine the location of the blob in the image. You can also have this function return a report containing 16 of the most

commonly used measurements, including the area, projection along the x-axis and y-axis, and perimeter of each blob.

- `imaqSelectParticles()`—This function selects information about blobs from the reports generated by `imaqGetParticleInfo()`. Blobs that do not meet the criteria you set are filtered from the reports.
- `imaqCalcCoeff()`—This function uses the reports from `imaqGetParticleInfo()` or `imaqSelectParticles()` to calculate 50 particle measurements.

Table 4-1 lists all of the measurements that `imaqCalcCoeff()` returns.

**Table 4-1.** Particle Measurements

| Measurement | Description |
|---|---|
| IMAQ_AREA | area of the particle in pixels |
| IMAQ_AREA_CALIBRATED | area of the particle in user-defined units |
| IMAQ_NUM_HOLES | number of holes within the particle |
| IMAQ_AREA_OF_HOLES | total area of the particle holes in pixels |
| IMAQ_TOTAL_AREA | area of the particle and its holes |
| IMAQ_IMAGE_AREA | area of the entire image in real-world units |
| IMAQ_PARTICLE_TO_IMAGE | ratio, expressed as a percent, of the surface area of a particle in relation to the image area |
| IMAQ_PARTICLE_TO_TOTAL | ratio, expressed as a percent, of the surface area of a particle in relation to the total area of the particle |
| IMAQ_CENTER_MASS_X | x-coordinate of the center of mass |
| IMAQ_CENTER_MASS_Y | y-coordinate of the center of mass |
| IMAQ_LEFT_COLUMN | left x-coordinate of the bounding rectangle |
| IMAQ_TOP_ROW | top y-coordinate of the bounding rectangle |
| IMAQ_RIGHT_COLUMN | right x-coordinate of the bounding rectangle |
| IMAQ_BOTTOM_ROW | bottom y-coordinate of bounding rectangle |
| IMAQ_WIDTH | width of bounding rectangle in user-defined units |
| IMAQ_HEIGHT | height of bounding rectangle in user-defined units |
| IMAQ_MAX_SEGMENT_LENGTH | length of longest horizontal line segment in a particle |

**Table 4-1.** Particle Measurements (Continued)

| Measurement | Description |
|---|---|
| IMAQ_MAX_SEGMENT_LEFT_COLUMN | leftmost x-coordinate of longest horizontal line segment in a particle |
| IMAQ_MAX_SEGMENT_TOP_ROW | y-coordinate of longest horizontal line segment |
| IMAQ_PERIMETER | length of the outer contour of the particle in user-defined units |
| IMAQ_PERIMETER_OF_HOLES | perimeter of all holes in user-defined units |
| IMAQ_SIGMA_X | sum of the x-coordinates for each pixel of the particle |
| IMAQ_SIGMA_Y | sum of the y-coordinates for each pixel of the particle |
| IMAQ_SIGMA_XX | sum of the squared x-coordinates for each pixel of the particle |
| IMAQ_SIGMA_YY | sum of the squared y-coordinates for each pixel of the particle |
| IMAQ_SIGMA_XY | sum of the product of the x-coordinate and y-coordinate for each pixel of the particle |
| IMAQ_PROJ_X | sum of the vertical segments in a particle |
| IMAQ_PROJ_Y | sum of the horizontal segments in a particle |
| IMAQ_INERTIA_XX | inertia matrix coefficient in XX |
| IMAQ_INERTIA_YY | inertia matrix coefficient in YY |
| IMAQ_INERTIA_XY | inertia matrix coefficient in XY |
| IMAQ_MEAN_H | mean length of horizontal segments |
| IMAQ_MEAN_V | mean length of vertical segments |
| IMAQ_MAX_INTERCEPT | length of the longest segment of the convex hull of the particle |
| IMAQ_MEAN_INTERCEPT | mean length of the chords in a particle perpendicular to its max intercept |
| IMAQ_ORIENTATION | direction of the major axis of a particle |
| IMAQ_EQUIV_ELLIPSE_MINOR | total length of the minor axis of the ellipse having the same area as the particle and a major axis equal to half the max intercept |

**Table 4-1.** Particle Measurements (Continued)

| Measurement | Description |
|---|---|
| IMAQ_ELLIPSE_MAJOR | total length of the major axis of the ellipse having the same area and perimeter as the particle in user-defined units |
| IMAQ_ELLIPSE_MINOR | total length of the minor axis of the ellipse having the same area and perimeter as the particle in user-defined units |
| IMAQ_ELLIPSE_RATIO | fraction of the length of the major axis to the length of the minor axis of the ellipse having the same area and perimeter as the particle in user-defined units |
| IMAQ_RECT_LONG_SIDE | length of the long side of a rectangle having the same area and perimeter as the particle in user-defined units |
| IMAQ_RECT_SHORT_SIDE | length of the short side of a rectangle having the same area and perimeter as the particle in user-defined units |
| IMAQ_RECT_RATIO | ratio of rectangle long side to rectangle short side |
| IMAQ_ELONGATION | max intercept/mean perpendicular intercept |
| IMAQ_COMPACTNESS | particle area/(height × width) |
| IMAQ_HEYWOOD | particle perimeter/perimeter of circle having same area as the particle |
| IMAQ_TYPE_FACTOR | complex factor relating the surface area to the moment of inertia |
| IMAQ_HYDRAULIC | particle area/particle perimeter |
| IMAQ_WADDLE_DISK | diameter of the disk having the same area as the particle in user-defined units |
| IMAQ_DIAGONAL | diagonal of an equivalent rectangle in user-defined units |

# Convert Pixel Coordinates to Real-World Coordinates

If you need to find the location of the center of mass or the bounding rectangle of the blobs in real-world units, use `imaqTransformPixelToRealWorld()`.

# 5

# Machine Vision

This chapter describes how to perform many common machine vision inspection tasks. The most common inspection tasks are detecting the presence or absence of parts in an image and measuring the dimensions of parts to see if they meet specifications.

Measurements are based on characteristic features of the object represented in the image. Image processing algorithms traditionally classify the type of information contained in an image as edges, surfaces and textures, or patterns. Different types of machine vision algorithms leverage and extract one or more types of information.

Edge detectors and derivative techniques—such as rakes, concentric rakes, and spokes—use edges represented in the image. They locate with high accuracy the position of the edge of an object in the image. For example, you can use the edge location to measure the width of the part (a technique called clamping). You can combine multiple edge locations to compute intersection points, projections, circles, or ellipse fits.

Pattern matching algorithms use edges and patterns. Pattern matching can locate with very high accuracy the position of fiducials or characteristic features of the part under inspection. Those locations can then be combined to compute lengths, angles, and other object measurements.

The robustness of your measurement relies on the stability of your image acquisition conditions. Sensor resolution, lighting, optics, vibration control, part fixture, and general environment are key components of the imaging setup. All the elements of the image acquisition chain directly affect the accuracy of the measurements.

Figure 5-1 illustrates the basic steps involved in performing machine vision. Diagram items enclosed with dashed lines are optional steps.

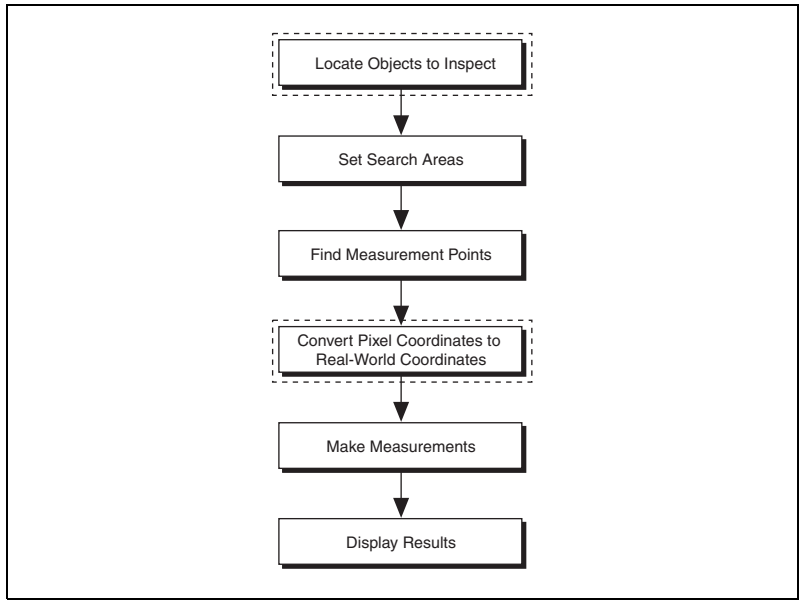



**Figure 5-1.** Steps to Performing Machine Vision

# Locate Objects to Inspect

In a typical machine vision application, you extract measurements from regions of interest rather than the entire image. To use this technique, the parts of the object you are interested in must always appear inside the regions of interest you define.

If the object under inspection is always at the same location and orientation in the images you need to process, defining regions of interest is simple. See the *Set Search Areas* section for information about selecting a region of interest.

Often, the object under inspection appears rotated or shifted in the image you need to process with respect to the reference image in which you located the object. When this occurs, the regions of interest (ROIs) need to shift and rotate with the parts of the object in which you are interested. In order for the ROIs to move with the object, you need to define a reference coordinate system relative to the object in the reference image. During the measurement process, the coordinate system moves with the object when it

appears shifted and rotated in the image you need to process. This coordinate system is referred to as the measurement coordinate system. The measurement methods automatically move the ROIs to the correct position using the position of the measurement coordinate system with respect to the reference coordinate system. For information about coordinate systems, see Chapter 13, *Dimensional Measurements*, of the *IMAQ Vision Concepts Manual*.

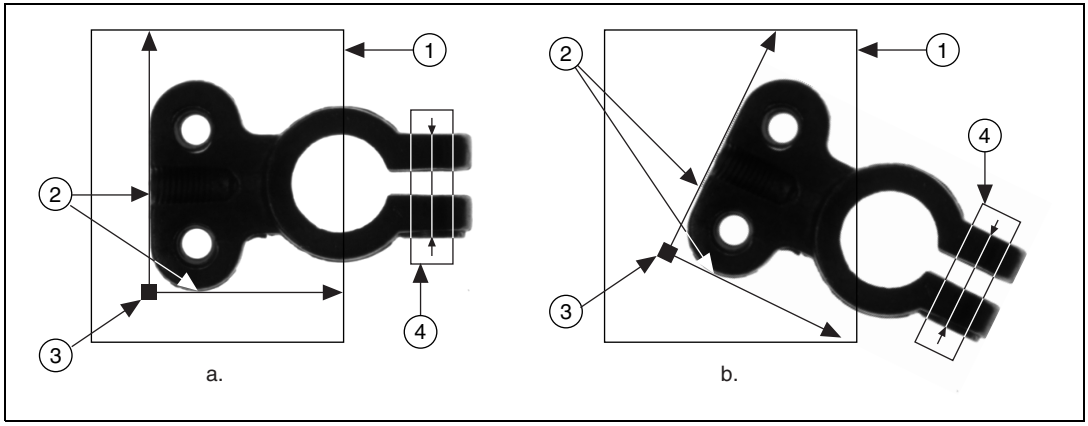
You can build a coordinate transform using edge detection or pattern matching. The output of the edge detection and pattern matching functions that build a coordinate system are the origin, angle, and axes direction of the coordinate system. Some machine vision functions take this output and adjust the regions of inspection automatically. You can also use these outputs to move the regions of inspection relative to the object programmatically.

## Using Edge Detection to Build a Coordinate Transform

You can build a coordinate transform using two edge detection techniques. Use `imaqFindTransformRect()` to define a coordinate system using one rectangular region. Use `imaqFindTransformRects()` to define a coordinate system using two independent rectangular regions. Follow the steps below to build a coordinate transform using edge detection:

**Note** To use this technique, the object cannot rotate more than ±65° in the image.

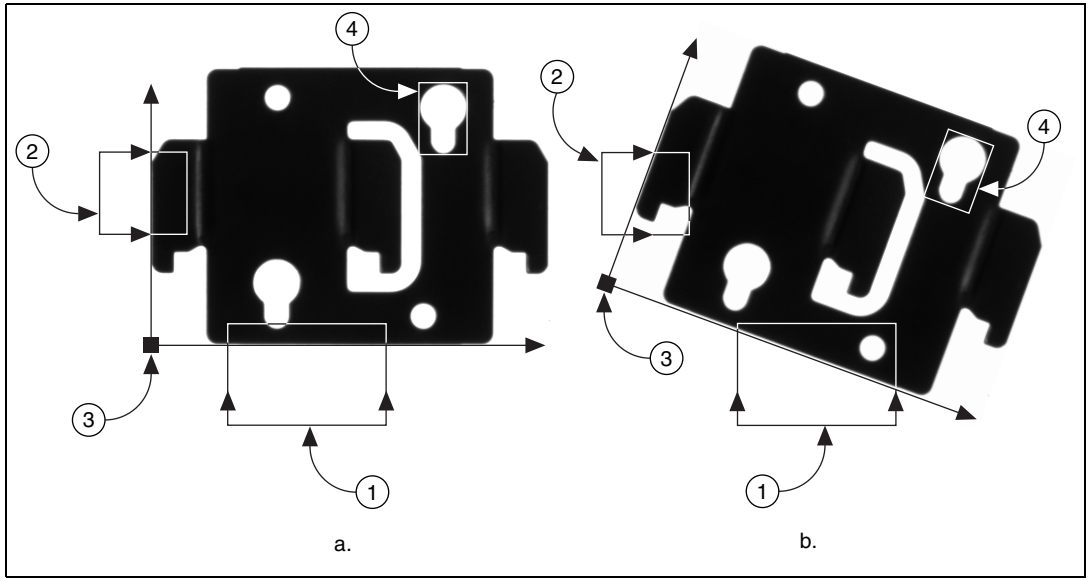
1. Specify one or two rectangular regions.
   a. If you use `imaqFindTransformRect()`, specify one rectangular ROI that includes part of two straight, nonparallel boundaries of the object, as shown in Figure 5-2. This rectangular region must be large enough to include these boundaries in all the images you want to inspect.




| | |
|---|---|
| 1 Search Area for the Coordinate System | 3 Origin of the Coordinate System |
| 2 Object Edges | 4 Measurement Area |

**Figure 5-2.** Coordinate Systems of a Reference Image and Inspection Image

b. If you use `imaqFindTransformRects()`, specify two rectangles, each containing one separate, straight boundary of the object, as shown in Figure 5-3. The boundaries cannot be parallel. The regions must be large enough to include the boundaries in all the images you want to inspect.




| | |
|---|---|
| 1 Primary Search Area | 3 Origin of the Coordinate System |
| 2 Secondary Search Area | 4 Measurement Area |

**Figure 5-3.** Locating Coordinate System Axes with Two Search Areas

2. Use the **options** parameter to choose the options you need to locate the edges on the object, the coordinate system axis direction, and the results that you want to overlay onto the image. Set the **options** parameter to NULL to use the default options.
3. Choose the mode for the function. To build a coordinate transform for the first time, set **mode** to IMAQ_FIND_REFERENCE. To update the coordinate transform in subsequent images, set this mode to IMAQ_UPDATE_TRANSFORM.

## Using Pattern Matching to Build a Coordinate Transform

You can build a coordinate transform using pattern matching. Use `imaqFindTransformPattern()` to define a coordinate system based on the location of a reference feature. Use this technique when the object under inspection does not have straight, distinct edges. Follow the steps below to build a coordinate reference system using pattern matching:

**Note** The object may rotate ±360° in the image using this technique if you use rotation-invariant pattern matching.

1. Define a template that represents the part of the object that you want to use as a reference feature. For more information about defining a template, see the *Find Measurement Points* section.
2. Define a rectangular search area in which you expect to find the template.
3. Use the **options** parameter to select your options for finding the pattern and the results that you want to overlay onto the image. When setting the `Mode` element, select `IMAQ_MATCH_ROTATION_INVARIANT` when you expect your template to appear rotated in the inspection images. Otherwise, select `IMAQ_MATCH_SHIFT_INVARIANT`. Set the **options** parameter to NULL to use the default options.
4. Choose the mode for the function. To build a coordinate transform for the first time, set **mode** to `IMAQ_FIND_REFERENCE`. To update the coordinate system in subsequent images, set **mode** to `IMAQ_UPDATE_TRANSFORM`.

## Choosing a Method to Build the Coordinate Transform

The following flowchart guides you through choosing the best method for building a coordinate transform for your application.

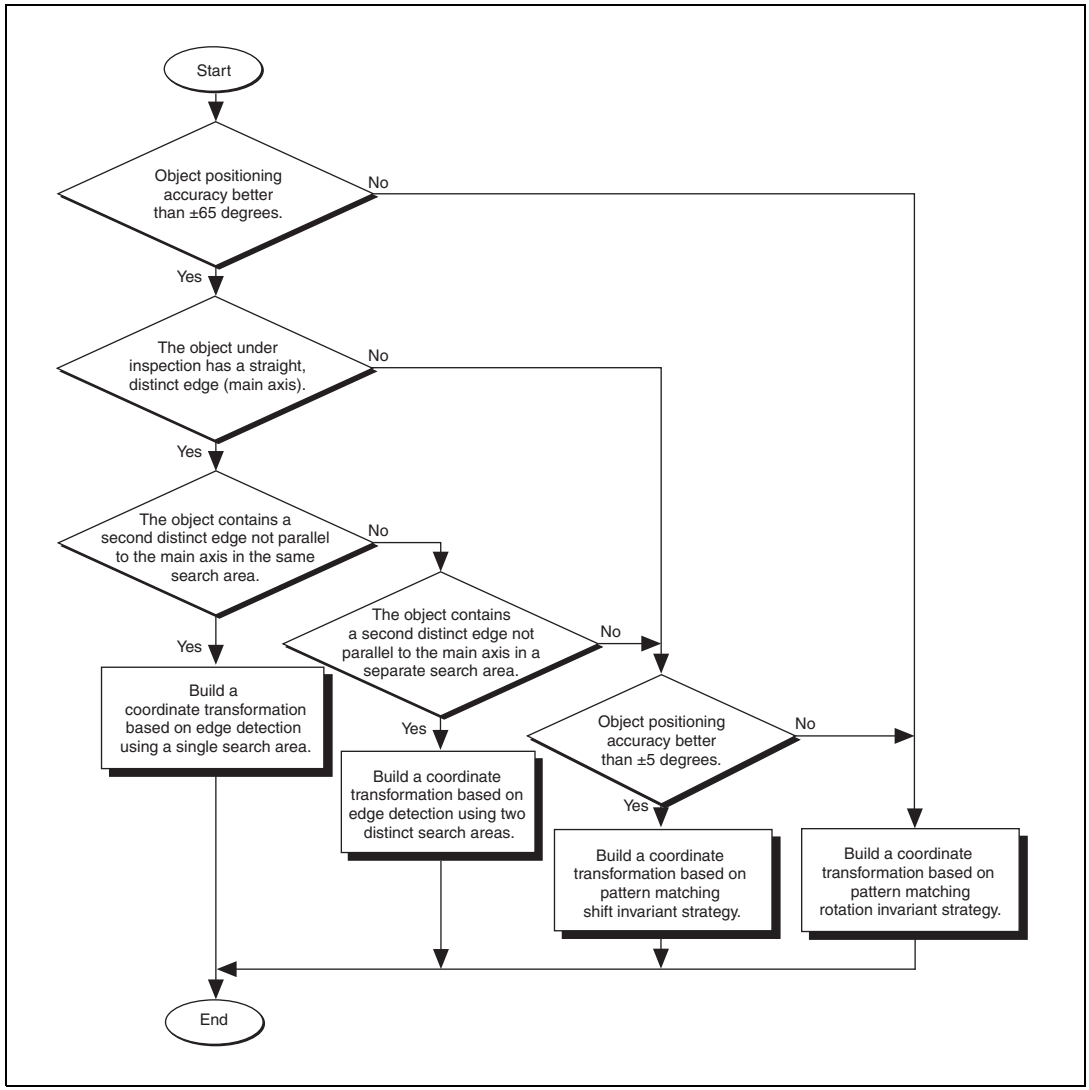



**Figure 5-4.** Building a Coordinate Transform

# Set Search Areas

Select regions of interest (ROIs) in your images to limit the areas in which you perform your processing and inspection. You can define ROIs interactively or programmatically.

## Interactively Defining Regions

Follow these steps to interactively define an ROI:

1. Use `imaqConstructROI()` to display an image and the tools palette in a window.
2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize and reposition the ROI until it specifies the area you want to process.
4. Click **OK** to output a descriptor of the region you selected. You can input the ROI descriptor into many analysis and processing functions.

You can also use `imaqSelectRect()` and `imaqSelectAnnulus()` to define regions of interest. Follow these steps to use these functions:

1. Call the function to display an image in a window. Only the tools specific to that function are available for you to use.
2. Select an ROI tool from the tools palette.
3. Draw an ROI on your image. Resize or reposition the ROI until it specifies the area you want to process.

Click **OK** to output a simple description of the ROI. You can use this description as an input for the following functions:

| ROI Selection Function | Measurement Function |
|---|---|
| `imaqSelectRect()` | `imaqFindPattern()`<br>`imaqClampMax()`<br>`imaqClampMin()`<br>`imaqFindEdge()` |
| `imaqSelectAnnulus()` | `imaqFindCircularEdge()`<br>`imaqFindConcentricEdge()` |

## Programmatically Defining Regions

When you have an automated application, you need to define regions of interest programmatically. You can programmatically define regions in two ways:

- Specify the contours of the ROI.
- Specify individual structures by providing basic parameters that describe the region you want to define. You can specify a rotated rectangle by providing the coordinates of the center, the width, the height, and the rotation angle. You can specify an annulus by providing the coordinates of the center, inner radius, outer radius, start angle, and end angle. You can specify a point by setting its x-coordinates and y-coordinates. You can specify a line by setting the coordinates of the start and end points.

See Chapter 3, *Grayscale and Color Measurements*, for more information about defining regions of interest.

# Find Measurement Points

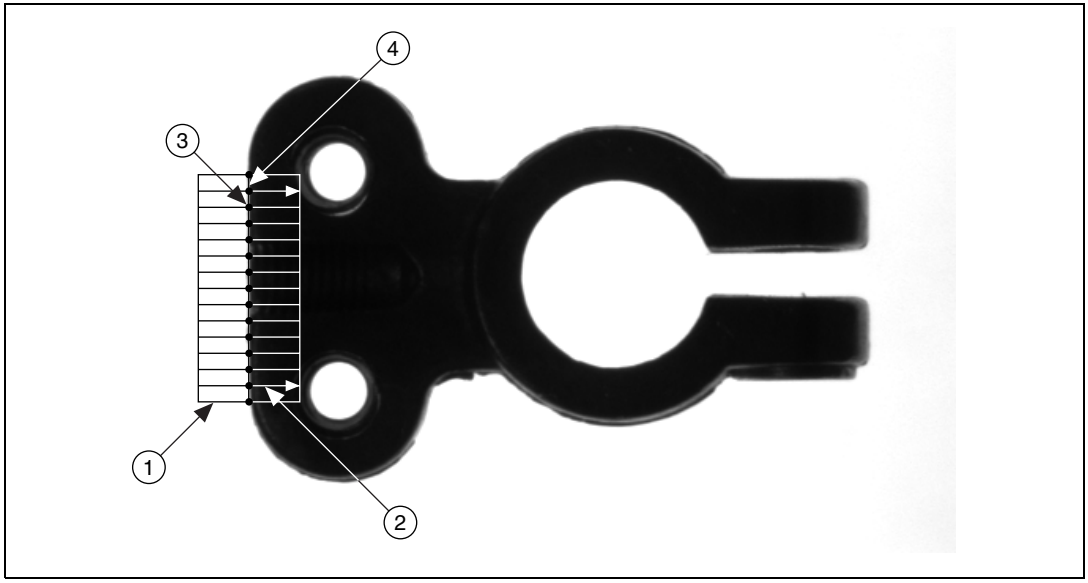
After you set regions of inspection, locate points within those regions on which you can base measurements. You can locate measurement points using edge detection, pattern matching, color pattern matching, and color location.

## Finding Features Using Edge Detection

Use the edge detection tools to identify and locate sharp discontinuities in an image. Discontinuities typically represent abrupt changes in pixel intensity values, which characterize the boundaries of objects.

### Finding Lines or Circles

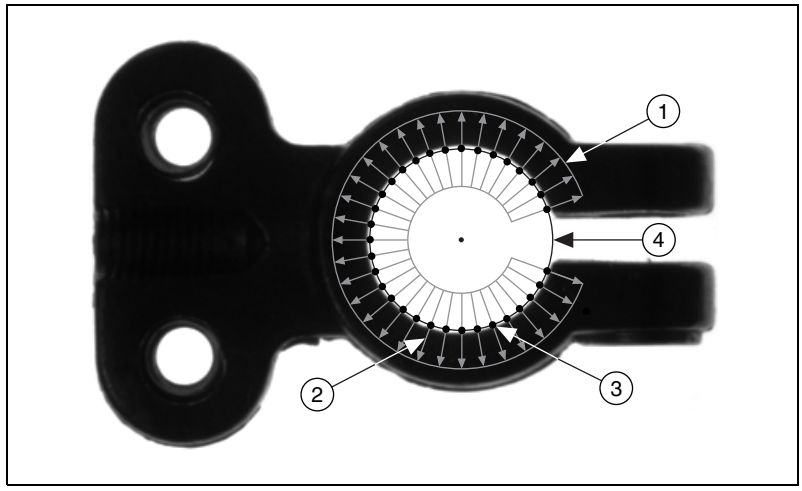
If you want to find points along the edge of an object and find a line describing the edge, use `imaqFindEdge()` and `imaqFindConcentricEdges()`. The `imaqFindEdge()` function finds edges based on rectangular search areas, as shown in Figure 5-5. The `imaqFindConcentricEdge()` function finds edges based on annular search areas.




| | |
|---|---|
| 1 Search Region | 3 Detected Edge Points |
| 2 Search Lines | 4 Line Fit to Edge Points |

**Figure 5-5.** Finding a Straight Feature

If you want to find points along a circular edge and find the circle that best fits the edge, as shown in Figure 5-6, use `imaqFindCircularEdge()`.




| | | | |
|---|---|---|---|
| 1 | Annular Search Region | 3 | Detected Edge Points |
| 2 | Search Lines | 4 | Circle Fit To Edge Points |

**Figure 5-6.** Finding a Circular Feature

These functions locate the intersection points between a set of search lines within the search region and the edge of an object. You can specify the search region using `imaqSelectRect()` or `imaqSelectAnnulus()`. Specify the separation between the lines that the functions use to detect edges. The functions determine the intersection points based on their contrast, width, and steepness. The software calculates a best-fit line with outliers rejected or a best-fit circle through the points it found. The functions return the coordinates of the edges found.

## Finding Edge Points Along One Search Contour

Use `imaqSimpleEdge()` or `imaqEdgeTool()` to find edge points along a contour. Using `imaqSimpleEdge()`, you can find the first edge, last edge, or all edges along the contour. Use `imaqSimpleEdge()` when your image contains little noise and the object and background are clearly differentiated. Otherwise, use `imaqEdgeTool()`.

These functions require you to input the coordinates of the points along the search contour. Use `imaqROIProfile()` to obtain the coordinates along the edge of each contour in an ROI. If you have a straight line, use `imaqGetPointsOnLine()` to obtain the points along the line instead of using an ROI.

These functions determine the edge points based on their contrast and slope. You can specify whether you want to find the edge points using subpixel accuracy.

## Finding Edge Points Along Multiple Search Contours

Use `imaqRake()`, `imaqSpoke()`, and `imaqConcentricRake()` to find edge points along multiple search contours. These functions behave like `imaqEdgeTool()` behaves, but they find edges on multiple contours. Pass in an ROI to define the search region for these functions.

The `imaqRake()` function works on a rectangular search region. The search lines are drawn parallel to the orientation of the rectangle. Control the number of search lines in the region by specifying the distance, in pixels, between each line. Specify the search direction as left to right or right to left for a horizontally oriented rectangle. Specify the search direction as top to bottom or bottom to top for a vertically oriented rectangle.

The `imaqSpoke()` function works on an annular search region, scanning the search lines that are drawn from the center of the region to the outer boundary and that fall within the search area. Control the number of lines in the region by specifying the angle, in degrees, between each line. Specify the search direction as either going from the center outward or from the outer boundary to the center.

The `imaqConcentricRake()` function works on an annular search region. The concentric rake is an adaptation of the Rake to an annular region. IMAQ Vision performs edge detection along search lines that occur in the search region and that are concentric to the outer circular boundary. Control the number of concentric search lines that are used for the edge detection by specifying the radial distance between the concentric lines in pixels. Specify the direction of the search as either clockwise or counterclockwise.

# Finding Points Using Pattern Matching

The pattern matching algorithms in IMAQ Vision measure the similarity between an idealized representation of a feature, called a template, and the feature that may be present in an image. A feature is defined as a specific pattern of pixels in an image. Pattern matching returns the location of the center of the template and the template orientation. Follow these generalized steps to find features in an image using pattern matching:

1. Define a reference or fiducial pattern in the form of a template image.
2. Use the reference pattern to train the pattern matching algorithm with `imaqLearnPattern()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchPattern()`.
5. Test the search algorithm on test images using `imaqMatchPattern()`.
6. Verify the results using a ranking method.

## Defining and Create Good Template Images

The selection of a good template image plays a critical part in obtaining good results. Because the template image represents the pattern that you want to find, make sure that all the important and unique characteristics of the pattern are well defined in the image.

Several factors are critical in creating a template image. These critical factors include symmetry, feature detail, positional information, and background information.

### Symmetry

A rotationally symmetric template is less sensitive to changes in rotation than one that is rotationally asymmetric. A rotationally symmetric template provides good positioning information but no orientation information.

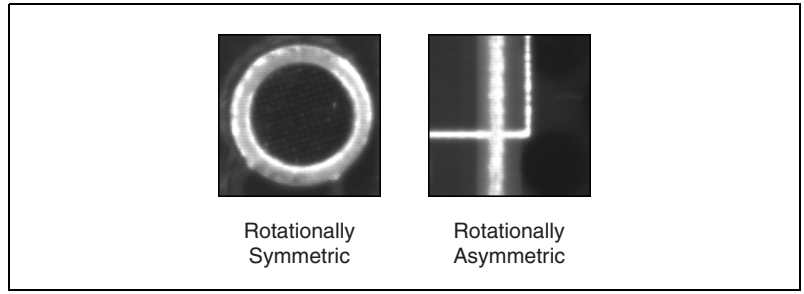



**Figure 5-7.** Symmetry

## Feature detail

A template with relatively coarse features is less sensitive to variations in size and rotation than a model with fine features. However, the model must contain enough detail to identify it.

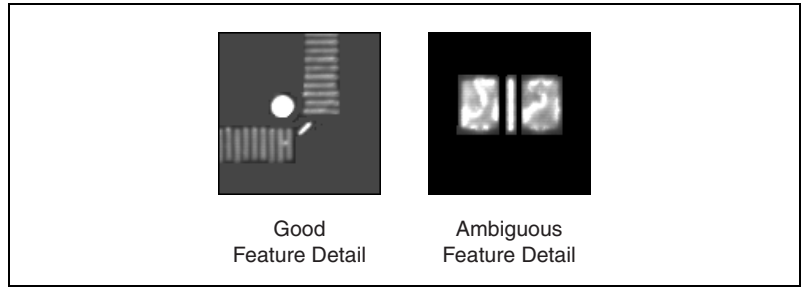



**Figure 5-8.** Feature Detail

## Positional information

A template with strong edges in both the x and y directions is easier to locate.

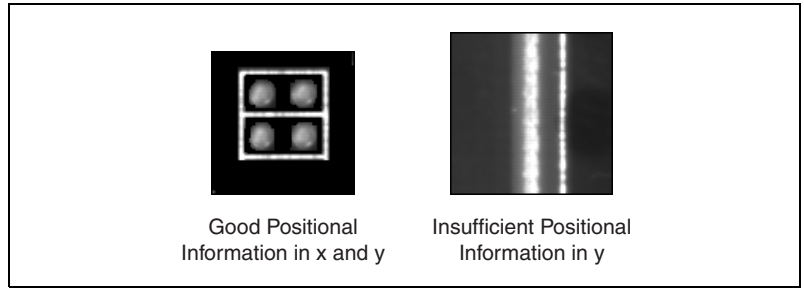



**Figure 5-9.** Positional Information

### Background information

Unique background information in a template improves search performance and accuracy.

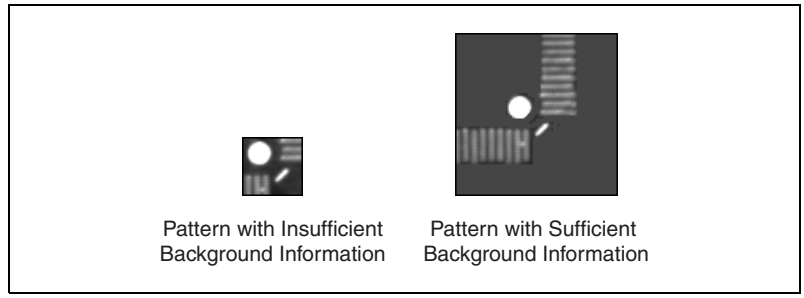



**Figure 5-10.** Background Information

## Training the Pattern Matching Algorithm

After you create a good template image, the pattern matching algorithm has to learn the important features of the template. Use `imaqLearnPattern()` to learn the template. The learning process depends on the type of matching that you expect to perform. If you do not expect the instance of the template in the image to rotate or change its size, then the pattern matching algorithm has to learn only those features from the template that are necessary for shift-invariant matching. However, if you want to match the template at any orientation, use rotation-invariant matching. Use the **learningMode** parameter of `imaqLearnPattern()` to specify which type of learning mode to use.

The learning process is usually time intensive because the algorithm attempts to find the optimum features of the template for the particular matching process. The learning mode you choose also affects the speed of the learning process. Learning the template for shift-invariant matching is faster than learning for rotation-invariant matching. You can also save time by training the pattern matching algorithm offline and then saving the template image with `imaqWriteVisionFile()`.

## Defining a Search Area

Two equally important factors define the success of a pattern matching algorithm: accuracy and speed. You can define a search area to reduce ambiguity in the search process. For example, if your image has multiple instances of a pattern and only one of them is required for the inspection task, the presence of additional instances of the pattern can produce

incorrect results. To avoid this, reduce the search area so that only the desired pattern lies within the search area.

The time required to locate a pattern in an image depends on both the template size and the search area. By reducing the search area or increasing the template size, you can reduce the required search time.

In many inspection applications, you have general information about the location of the fiducial. You should use this information to define a search area. For example, in a typical component placement application, each printed circuit board (PCB) being tested may not be placed in the same location with the same orientation. The location of the PCB in various images can move and rotate within a known range of values, as illustrated in Figure 5-11. Figure 5-11a shows the template used to locate the PCB in the image. Figure 5-11b shows an image containing a PCB with a fiducial you want to locate. Notice the search area around the fiducial. If you know before the matching process begins that the PCB can shift or rotate in the image within a fixed range (as shown in Figure 5-11c and Figure 5-11d, respectively), then you can limit the search for the fiducial to a small region of the image.

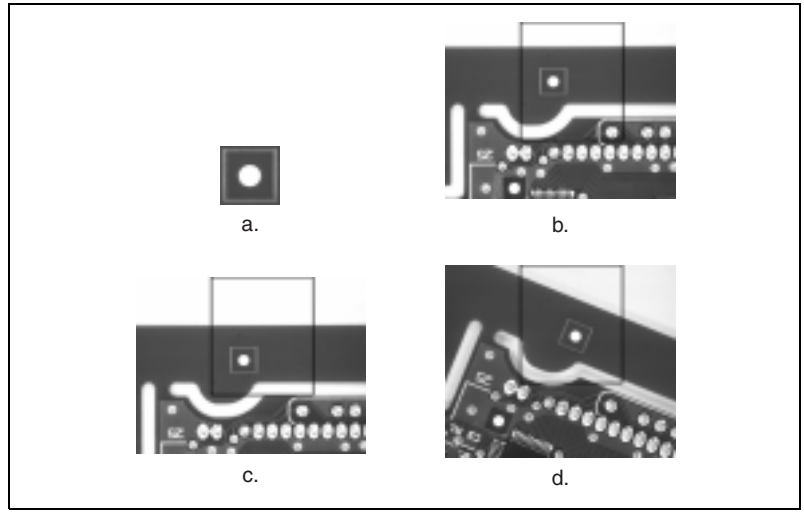



**Figure 5-11.** Selecting a Search Area for Grayscale Pattern Matching

## Setting Matching Parameters and Tolerances

Every pattern matching algorithm makes assumptions about the images and pattern matching parameters used in machine vision applications. These assumptions work for a high percentage of the applications.

However, there may be applications in which the assumptions used in the algorithm are not optimal. Knowing your particular application and the images you want to process is useful in selecting the pattern matching parameters. The following are parameters the influence the IMAQ Vision pattern matching algorithm.

## Match Mode

You can set the match mode to control how the pattern matching algorithm treats the template at different orientations. If you expect the orientation of valid matches to vary less than ±5° from the template, set the `mode` element of the **options** parameter to `IMAQ_MATCH_SHIFT_INVARIANT`. Otherwise, set the mode element to `IMAQ_MATCH_ROTATION_INVARIANT`. Shift-invariant matching is faster than rotation-invariant matching.

## Minimum Contrast

You can set the minimum contrast to potentially increase the pattern matching algorithm's speed. Contrast is the difference between the smallest and largest pixel values in a region. The pattern matching algorithm ignores all image regions where contrast values fall beneath a set minimum contrast value. If the search image has high contrast but contains some low contrast regions, you can set a high minimum contrast value. By using a high minimum contrast value, you exclude all areas in the image with low contrast, significantly reducing the region in which the pattern matching algorithm must search. If the search image has low contrast throughout, set a low minimum contrast parameter to ensure that the pattern matching algorithm looks for the template in all regions of the image. Use the `minContrast` element of the `imaqMatchPattern()` **options** parameter to set the minimum contrast.

## Rotation Angle Ranges

For matching objects that may rotate in the image, the pattern matching algorithm, by default, allows any orientation between 0° to 360°. If the pattern rotation in your application is restricted to a certain range (for example, from –15° to 15°), you can provide this information to the pattern matching algorithm by setting the `angleRanges` element of the `imaqMatchPattern()` **options** parameter. This information improves your search time because the pattern matching algorithm looks for the pattern at fewer angles.

## Testing the Search Algorithm on Test Images

To determine if your selected template or reference pattern is appropriate for your machine vision application, test the template on a few test images by using `imaqMatchPattern()`. These test images should reflect the images generated by your machine vision application during true operating conditions. If the pattern matching algorithm locates the reference pattern in all cases, you have selected a good template. Otherwise, refine the current template, or select a better template until both training and testing are successful.

## Using a Ranking Method to Verify Results

The manner in which you interpret the pattern matching algorithm depends on your application. For typical alignment applications, such as finding a fiducial on a wafer, the most important information is the position and location of the best match. Use the `position` and `corner` elements of the `Pattern Match` structure to get the position and the bounding rectangle of a match.

In inspection applications, such as optical character verification (OCV), the score of the best match is more useful. The score of a match returned by the pattern matching algorithm is an indicator of the closeness between the original pattern and the match found in the image. A high score indicates a very close match, while a low score indicates a poor match. The score can be used as a gauge to determine whether a printed character is acceptable. Use the `score` element of the `Pattern Match` structure to get the score corresponding to a match.

# Finding Points Using Color Pattern Matching

Color pattern matching algorithms provide a quick way to locate objects when color is present. Use color pattern matching if:

- The object you want to locate has color information that is very different from the background, and you want to find a very precise location of the object in the image.
- The object to locate has grayscale properties that are very difficult to characterize or that are very similar to other objects in the search image. In such cases, grayscale pattern matching can give inaccurate results. If the object has color information that differentiates it from the other objects in the scene, color provides the machine vision software with the additional information to locate the object.

Color pattern matching returns the location of the center of the template and the template orientation. Follow these general steps to find features in an image using color pattern matching:

1. Define a reference or fiducial pattern in the form of a template image.
2. Use the reference pattern to train the color pattern matching algorithm with `imaqLearnColorPattern()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the `featureMode` element of the `imaqMatchColorPattern()` **options** parameter to `IMAQ_COLOR_AND_SHAPE_FEATURES`.
5. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchColorPattern()`.
6. Test the search algorithm on test images using `imaqMatchColorPattern()`.
7. Verify the results using a ranking method.

## Defining and Creating Good Color Template Images

The selection of a good template image plays a critical part in obtaining accurate results with the color pattern matching algorithm. Because the template image represents the color and the pattern that you want to find, make sure that all the important and unique characteristics of the pattern are well defined in the image.

Several factors are critical in creating a template image. These critical factors include color information, symmetry, feature detail, positional information, and background information.

### Color Information

A template with colors that are unique to the pattern provides better results than a template that contains many colors, especially colors found in the background or other objects in the image.

### Symmetry

A rotationally symmetric template in the luminance plane is less sensitive to changes in rotation than one that is rotationally asymmetric.

### Feature Detail

A template with relatively coarse features is less sensitive to variations in size and rotation than a model with fine features. However, the model must contain enough detail to identify it.

### Positional Information

A template with strong edges in both the x and y directions is easier to locate.

### Background Information

Unique background information in a template improves search performance and accuracy during the grayscale pattern matching phase. This requirement could conflict with the “color information” requirement because background colors may not be desirable during the color location phase. Avoid this problem by choosing a template with sufficient background information for grayscale pattern matching while specifying the exclusion of the background color during the color location phase. Refer to the *Training the Color Pattern Matching Algorithm* section for more information about how to ignore colors.

## Training the Color Pattern Matching Algorithm

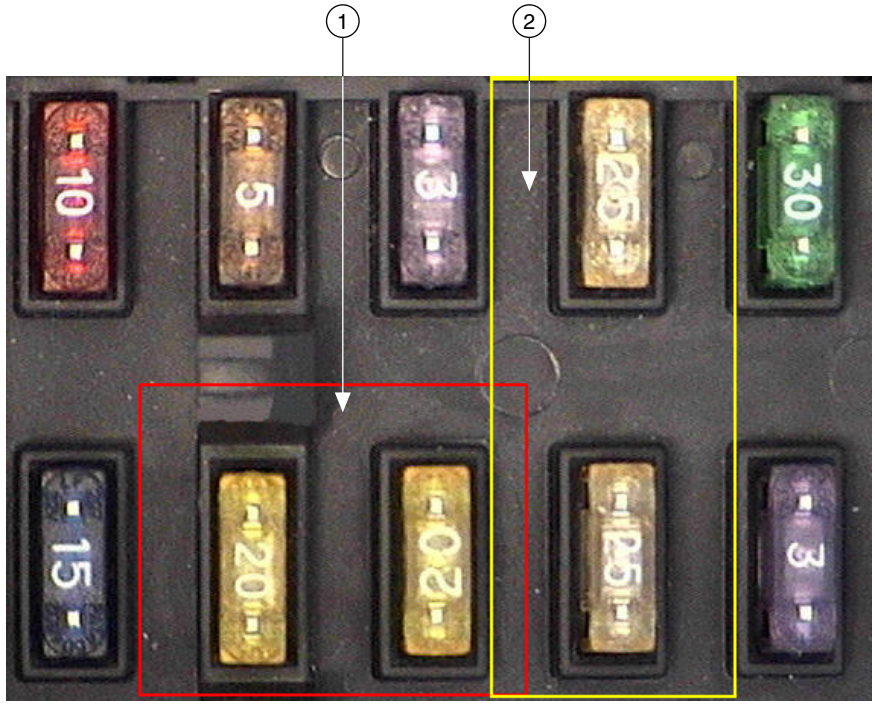
After you have created a good template image, the color pattern matching algorithm learns the important features of the template. Use `imaqLearnColorPattern()` to learn the template.The learning process depends on the type of matching that you expect to perform. By default, the color pattern matching algorithm learns only those features from the template that are necessary for shift-invariant matching. However, if you want to match the template at any orientation, the learning process must consider the possibility of arbitrary orientations. Use the `learnMode` element of the `imaqLearnColorPattern()` **options** parameter to specify which type of learning mode to use.

Exclude colors in the template that you are not interested in using during the search phase. Typically you should ignore colors that either belong to the background of the object or are not unique to the template, reducing the potential for incorrect matches during the color location phase. You can ignore certain predefined colors using the `ignoreMode` element of the **options** parameter. To ignore other colors, first learn the colors to ignore using `imaqLearnColor()`. Then set the `colorsToIgnore` element of the **options** parameter to the resulting `ColorInformation` structure from `imaqLearnColor()`.

The training or learning process is time-intensive because the algorithm attempts to find optimal features of the template for the particular matching process. However, you can train the pattern matching algorithm offline, and save the template image using `imaqWriteVisionFile()`.

## Defining a Search Area

Two equally important factors define the success of a color pattern matching algorithm—accuracy and speed. You can define a search area to reduce ambiguity in the search process. For example, if your image has multiple instances of a pattern and only one instance is required for the inspection task, the presence of additional instances of the pattern can produce incorrect results. To avoid this, reduce the search area so that only the desired pattern lies within the search area. For example, in the fuse box inspection example use the location of the fuses to be inspected to define the search area. Because the inspected fuse box may not be in the exact location or have the same orientation in the image as the previous one, the search area you define should be large enough to accommodate these variations in the position of the box. Figure 5-12 shows how search areas can be selected for different objects.




1 Search Area for 20 Amp Fuses 2 Search Area for 25 Amp Fuses

**Figure 5-12.** Selecting a Search Area for Color Pattern Matching

The time required to locate a pattern in an image depends on both the template size and the search area. By reducing the search area or increasing the template size, you can reduce the required search time.

## Setting Matching Parameters and Tolerances

Every color pattern matching algorithm makes assumptions about the images and color pattern matching parameters used in machine vision applications. These assumptions work for a high percentage of the applications.

In some applications, the assumptions used in the algorithm are not optimal. In such cases, you must modify the color pattern matching parameters. Knowing your particular application and the images you want to process is useful in selecting the pattern matching parameters. Use the **options** parameter of `imaqMatchColorPattern()` to set these elements.

The following are some elements in the IMAQ Vision pattern matching algorithm and how they influence pattern matching: color sensitivity, search strategy, color score weight, ignore background colors, minimum contrast, and rotation angle ranges.

## Color Sensitivity

Use the `sensitivity` element to control the granularity of the color information in the template image. If the background and objects in the image contain colors that are very close to colors in the template image, use a higher color sensitivity setting. A higher sensitivity setting distinguishes colors with very close hue values. Three color sensitivity settings are available in IMAQ Vision: `IMAQ_SENSITIVITY_LOW`, `IMAQ_SENSITIVITY_MED`, and `IMAQ_SENSITIVITY_HIGH`. Use the default Low setting if the colors in the template are very different from the colors in the background or other objects that you are not interested in. Increase the color sensitivity settings as the color differences decrease. For more information on color sensitivity, see Chapter 14, *Color Inspection*, of the *IMAQ Vision Concepts Manual*.

## Search Strategy

Use the `strategy` element to optimize the speed of the color pattern matching algorithm. The search strategy controls the step size, sub-sampling factor, percentage of color information used from the template.

Choose from four different strategies:

- `IMAQ_VERY_AGGRESSIVE`—Uses the largest step size, the most sub-sampling and only the dominant color from the template to search for the template. Use this strategy when the color in the template is almost uniform, the template is well contrasted from the background and there is a good amount of separation between different occurrences of the template in the image. This strategy is the fastest way to find templates in an image.
- `IMAQ_AGGRESSIVE`—Uses a large step size, a lot of sub-sampling, and little of the color information from the template.
- `IMAQ_BALANCED`—Uses values in between the `IMAQ_AGGRESSIVE` and `IMAQ_CONSERVATIVE` strategies.
- `IMAQ_CONSERVATIVE`—Uses a very small step size, a sub-sampling factor of two, and all the color information present in the template. This strategy is the most reliable method to look for a template in any image at potentially reduced speed.

**Note** Use the `IMAQ_CONSERVATIVE` strategy if you have multiple targets located very close to each other in the image.

Decide on the best strategy by experimenting with the different options.

### Color Score Weight

When you search for a template using both color and shape information, the color and shape scores generated during the match process are combined to generate the final color pattern matching score. The color score weight determines the contribution of the color score to the final color pattern matching score. If the template's color information is superior to its shape information, set the weight higher. For example, if you set `colorWeight` to 1000, the algorithm finds each match by using both color and shape information and then ranks the matches based entirely on their color scores. If you set `colorWeight` to 0, the matches are ranked based entirely on their shape scores.

### Minimum Contrast

Use the `minContrast` element to increase the color pattern matching algorithm's speed. The color pattern matching algorithm ignores all image regions where grayscale contrast values fall beneath a set minimum contrast value. See the *Setting Matching Parameters and Tolerances* section of this chapter for more information about minimum contrast.

### Rotation Angle Ranges

If you know that the pattern rotation is restricted to a certain range (for example, between –15° to 15°), provide this restriction information to the pattern matching algorithm by setting the `angleRanges` element. This information improves your search time because the color pattern matching algorithm looks for the pattern at fewer angles. See Chapter 12, *Pattern Matching*, in the *IMAQ Vision Concepts Manual* for more information on pattern matching.

## Testing the Search Algorithm on Test Images

To determine if your selected template or reference pattern is appropriate for your machine vision application, test the template on a few test images by using `imaqMatchColorPattern()`. These test images should reflect the images generated by your machine vision application during true operating conditions. If the pattern matching algorithm locates the

reference pattern in all cases, you have selected a good template. Otherwise, refine the current template, or select a better template until both training and testing are successful.

## Finding Points Using Color Location

Color location algorithms provide a quick way to locate regions in an image with specific colors.

Use color location when your application:

- Requires the location and the number of regions in an image with their specific color information
- Relies on the cumulative color information in the region, instead of the color arrangement in the region
- Does not require the orientation of the region
- Does not always require the location with sub-pixel accuracy
- Does not require shape information for the region

Follow these general steps to find features in an image using color location:

1. Define a reference pattern in the form of a template image.
2. Use the reference pattern to train the color location algorithm with `imaqLearnColorPattern()`.
3. Define an image or an area of an image as the search area. A small search area reduces the time to find the features.
4. Set the `featureMode` element of the `imaqMatchColorPattern()` **options** parameter to `IMAQ_COLOR_FEATURES`.
5. Set the tolerances and parameters to specify how the algorithm operates at run time using the **options** parameter of `imaqMatchColorPattern()`.
6. Test the color location algorithm on test images using `imaqMatchColorPattern()`.
7. Verify the results using a ranking method.

You can save the template image using `imaqWriteVisionFile()`.

# Convert Pixel Coordinates to Real-World Coordinates

The measurement points you located with edge detection and pattern matching are in pixel coordinates. If you need to make measurements using real-world units, use `imaqTransformPixelToRealWorld()` to convert the pixel coordinates into real-world units.

# Make Measurements

You can make different types of measurements either directly from the image or from points that you detect in the image.

## Distance Measurements

Use the following functions to make distance measurements for your inspection application.

Clamp functions measure the separation between two edges in a rectangular search region. First, clamp functions detect points along the two edges using the rake function. Then, they compute the distance between the points detected on the edges along each search line of the rake and return the largest or smallest distance. The `imaqSelectRect()` function generates a valid input search region for these functions. You also need to specify the parameters for edge detection and the separation between the search lines that you want to use within the search region to find the edges. These functions work directly on the image under inspection, and they output the coordinates of all the edge points that they find. The following list describes the available clamp functions:

- `imaqClampMax()`—Measures the largest separation between two edges in a rectangular search region.
- `imaqClampMin()`—Finds the smallest separation between two edges.

Use `imaqGetDistance()` to compute the distances between two points, such as consecutive pairs of points in an array of points. You can obtain these points from the image using any one of the feature detection methods described in the *Find Measurement Points* section of this chapter.

## Analytic Geometry Measurements

Use the following functions to make geometrical measurements from the points you detect in the image:

- `imaqFitLine()`—Fits a line to a set of points and computes the equation of the line.
- `imaqFitCircle()`—Fits a circle to a set of at least three points and computes its area, perimeter and radius.
- `imaqFitEllipse()`—Fits an ellipse to a set of at least six points and computes its area, perimeter, and the lengths of its major and minor axis.
- `imaqGetIntersection()`—Finds the intersection point of two lines specified by their start and end points.
- `imaqGetAngle()`—Finds the smaller angle between two lines.
- `imaqGetPerpendicularLine()`—Finds the perpendicular line from a point to a line and computes the perpendicular distance between the point and the line.
- `imaqGetBisectingLine()`—Finds the line that bisects the angle formed by two lines.
- `imaqGetMidLine()`—Finds the line that is midway between a point and a line and is parallel to the line.
- `imaqGetPolygonArea()`—Calculates the area of a polygon specified by its vertex points.

## Instrument Reader Measurements

You can make measurements based on the values obtained by meter, LCD, and barcode readers.

Use `imaqGetMeterArc()` to calibrate a meter or gauge that you want to read. The `imaqGetMeterArc()` function calibrates the meter using one of two modes. The `IMAQ_METER_ARC_ROI` mode uses the initial position and the full-scale position of the needle. When using this mode, the function calculates the position of the base of the needle and the arc traced by the tip of the needle. The `IMAQ_METER_ARC_POINTS` mode calibrates the meter using three points on the meter: the base of the needle, the tip of the needle at its initial position, and the tip of the needle at its full-scale position. When using this mode, the function calculates the position of the points along the arc covered by the tip of the needle. Use `imaqReadMeter()` to read the position of the needle using the base of the needle and the array of points on the arc traced by the tip of the needle.

Use `imaqFindLCDSegments()` to calculate the regions of interest around each digit in an LCD or LED. To find the area of each digit, all the segments of the indicator must be activated. Use `imaqReadLCD()` to read multiple digits of an LCD or LED.

Use `imaqReadBarcode()` to read values encoded in 1D barcodes. First, specify a region of interest that encloses the barcode information, and specify the type of barcode. Then, read the barcode.

# Display Results

You can display the results obtained at various stages of you inspection process on the window that displays your inspection image. You can do this by overlaying information on an image. The software attaches the information that you want to overlay to the image, but it does not modify the image. The overlay appears every time you display the image in an external window.

Use the following functions to overlay search regions, inspection results, and other information, such as text and bitmaps.

- `imaqOverlayPoints()`—Overlays points on an image. Specify a point by its x-coordinate and y-coordinate.
- `imaqOverlayLine()`—Overlays a line on an image. Specify a line by its start and end points.
- `imaqOverlayRect()`—Overlays a rectangle on an image.
- `imaqOverlayOval()`—Overlays an oval or a circle on the image.
- `imaqOverlayArc()`—Overlays an arc on the image.
- `imaqOverlayMetafile()`—Overlays a metafile on the image.
- `imaqOverlayText()`—Overlays text on an image.
- `imaqOverlayROI()`—Overlays an ROI on an image.
- `imaqOverlayClosedContour()`—Overlays a closed contour on an image.
- `imaqOverlayOpenContour()`—Overlays an open contour on an image.

To use these functions, pass in the image on which you want to overlay information and the information that you want to overlay. You can select the color of overlays by using the above functions.

You can configure the following processing functions to overlay different types of information on the inspection image:

- `imaqFindEdge()`
- `imaqFindCircularEdge()`
- `imaqFindConcentricEdge()`
- `imaqClampMax()`
- `imaqClampMin()`
- `imaqFindPattern()`
- `imaqCountObjects()`
- `imaqFindTransformRect()`
- `imaqFindTransformRects()`
- `imaqFindTransformPattern()`

You can overlay the following information with all the above functions except `imaqFindPattern()`:

- The search area input into the function.
- The search lines used for edge detection.
- The edges detected along the search lines
- The result of the function.

With `imaqFindPattern()`, you can overlay the search area and the result. Select the information you want to overlay by setting the element that corresponds to the information type to TRUE in the **options** input parameter.

Use `imaqClearOverlay()` to clear any previous overlay information from the image. Use `imaqWriteVisionFile()` to save an image with its overlay information to a file. You can read the information from the file into an image using `imaqReadVisionFile()`. As with calibration information, overlay information is removed from an image when the image size or orientation changes.

# 6

# Calibration

This chapter describes how to calibrate your imaging system, save calibration information, and attach calibration information to an image.

After you set up your imaging system, you may want to calibrate your system. If your imaging setup is such that the camera axis is perpendicular or nearly perpendicular to the object under inspection and your lens has no distortion, use simple calibration. With simple calibration, you do not need to learn a template. Instead, you define the distance between pixels in the horizontal and vertical directions using real-world units.

If your camera axis is not perpendicular to the object under inspection or your lens is distorted, use perspective and nonlinear distortion calibration to calibrate your system.

## Perspective and Nonlinear Distortion Calibration

Perspective errors and lens aberrations cause images to appear distorted. This distortion misplaces information in an image, but it does not necessarily destroy the information in the image. Calibrate your imaging system if you need to compensate for perspective errors or nonlinear lens distortion.

Follow these general steps to calibrate your imaging system:

1. Define a calibration template.
2. Define a reference coordinate system.
3. Learn the calibration information.

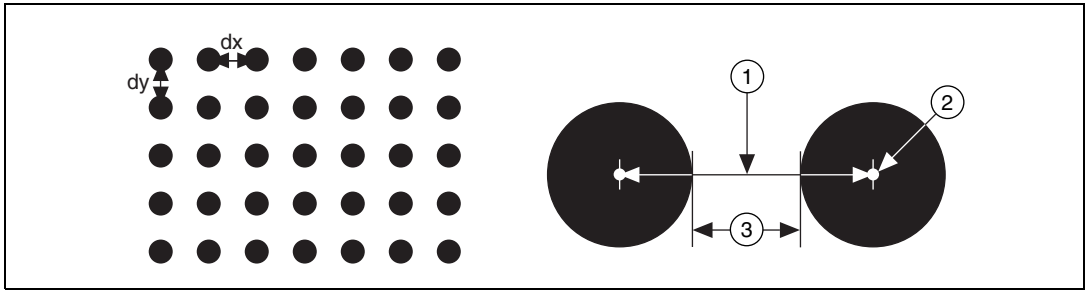
After you calibrate your imaging setup, you can attach the calibration information to an image. See the *Attach Calibration Information* section of this chapter for more information. Depending on your needs, you can either apply the calibration information to convert pixel coordinates to real-world coordinates without correcting the image, or you can create a distortion-free image by correcting the image for perspective errors and lens aberrations. See Chapter 4, *Blob Analysis*, and Chapter 5, *Machine Vision*, for more information about applying calibration information before making measurements.

## Defining a Calibration Template

You can define a calibration template by supplying an image of a grid or providing a list of pixel coordinates and their corresponding real-world coordinates. This section discusses the grid method in detail.

A calibration template is a user-defined grid of circular dots. As shown in Figure 6-1, the grid has constant spacings in the *x* and *y* directions. You can use any grid, but follow these guidelines for best results:

- The displacement in the *x* and *y* directions should be equal ($dx = dy$).
- The dots should cover the entire desired working area.
- The radius of the dots should be 6–10 pixels.
- The center-to-center distance between dots should range from 18 to 32 pixels, as shown in Figure 6-1.
- The minimum distance between the edges of the dots should be 6 pixels, as shown in Figure 6-1.




1 Center-to-Center Distance 2 Center of Grid Dots 3 Distance Between Dot Edges

**Figure 6-1.** Defining a Calibration Grid

**Note** You can use the calibration grid installed with IMAQ Vision at **Start»Programs»National Instruments»Vision»Documentation»Calibration Grid**. The dots have radii of 2 mm and center-to-center distances of 1 cm. Depending on your printer, these measurements may change by a fraction of a millimeter. You can purchase highly accurate calibration grids from optics suppliers, such as Edmund Industrial Optics.

## Defining a Reference Coordinate System

To express measurements in real-world units, you need to define a coordinate system in the image of the grid. Use the `CoordinateSystem` structure to define a coordinate system by its origin, angle, and axis direction.

The origin, expressed in pixels, defines the center of your coordinate system. The angle specifies the orientation of your coordinate system with respect to the angle of the topmost row of dots in the grid image. The calibration procedure automatically determines the direction of the horizontal axis in the real world. The vertical axis direction can either be indirect, as shown in Figure 6-2a, or direct, as shown in Figure 6-2b.

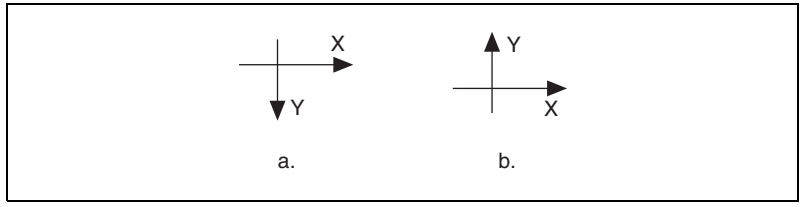
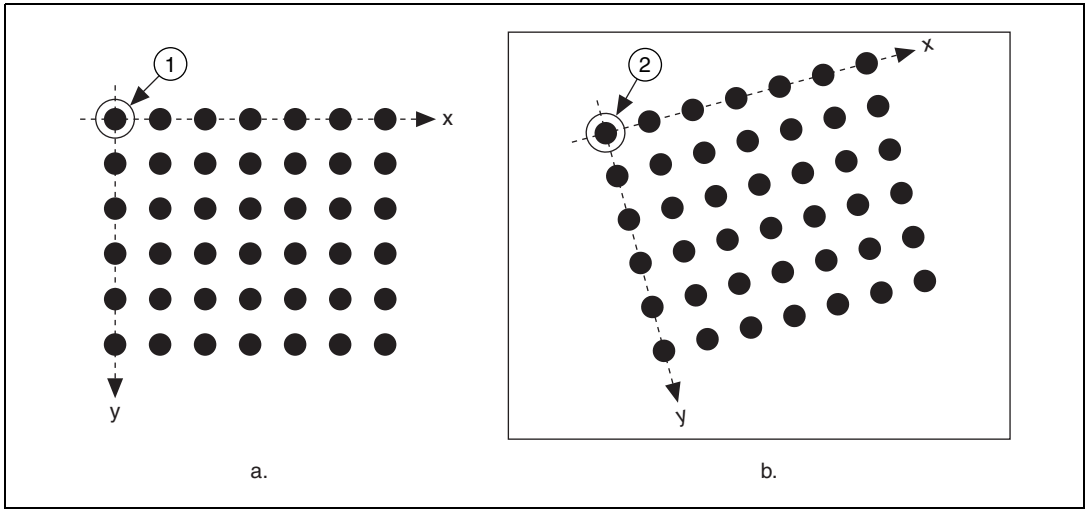



**Figure 6-2.** Axis Direction in the Image Plane

If you do not specify a coordinate system, the calibration process defines a default coordinate system. If you specify a grid for the calibration process, the software defines the following default coordinate system, as shown in Figure 6-3:

1. The origin is placed at the center of the left, topmost dot in the calibration grid.
2. The angle is set to 0°. This aligns the x-axis with the first row of dots in the grid, as shown in Figure 6-3b.
3. The axis direction is set to *indirect*. This aligns the y-axis to the first column of the dots in the grid, as shown in Figure 6-3b.




1 Origin of a Calibration Grid in the Real World 2 Origin of the Same Calibration Grid in an Image

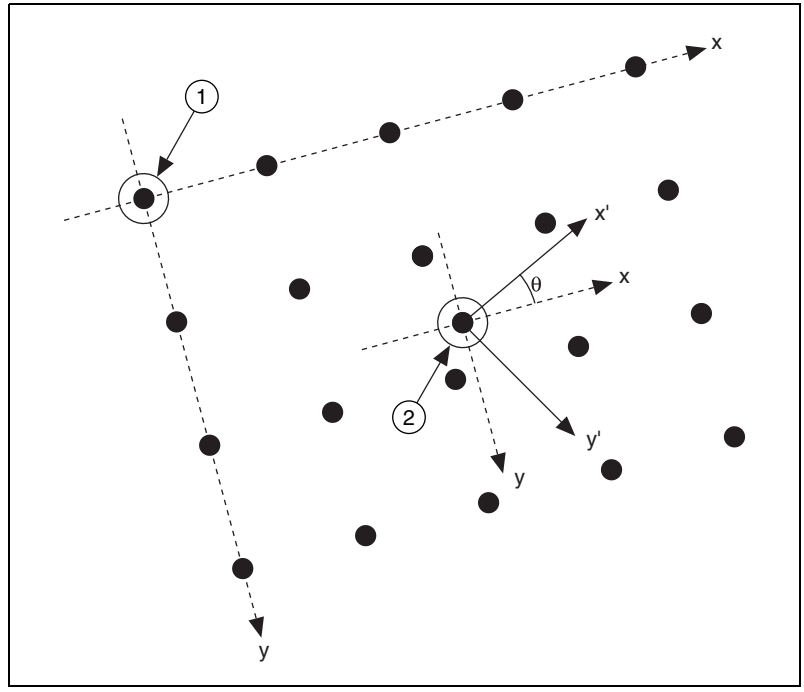
**Figure 6-3.** A Calibration Grid and an Image of the Grid

**Note** If you specify a list of points instead of a grid for the calibration process, the software defines a default coordinate system, as follows:

1. The origin is placed at the point in the list with the lowest x-coordinate value and then the lowest y-coordinate value.
2. The angle is set to 0°.
3. The axis direction is set to *indirect*.

If you define a coordinate system yourself, carefully consider the needs of your application. Remember the following:

- Express the origin in pixels. Always choose an origin location that lies within the calibration grid so that you can convert the location to real-world units.
- Specify the angle as the angle between the x-axis of the new coordinate system (x') and the top row of dots (x), as shown in Figure 6-4. If your imaging system exhibits nonlinear distortion, you cannot visualize the angle as you can in Figure 6-4 because the dots do not appear in straight lines.




1 Default Origin in a Calibration Grid Image 2 User-Defined Origin

**Figure 6-4.** Defining a Coordinate System

## Learning Calibration Information

After you define a calibration grid and reference axis, acquire an image of the grid using the current imaging setup. For information about acquiring images, see the *Acquire or Read an Image* section of Chapter 2, *Getting Measurement-Ready Images*. The grid does not need to occupy the entire image. You can choose a region within the image that contains the grid. After you acquire an image of the grid, learn the calibration information by inputting the image of the grid into `imaqLearnCalibrationGrid()`.

**Note** If you want to specify a list of points instead of a grid, use `imaqLearnCalibrationPoints()` to learn the calibration information. Use the `CalibrationPoints` structure to specify the pixel to real-world mapping.

## Specifying Scaling Factors

Scaling factors are the real-world distances between the dots in the calibration grid in the x and y directions and the units in which the distances are measured. Use the `GridDescriptor` structure to specify the scaling factors.

## Choosing a Region of Interest

Define a learning region of interest (ROI) during the learning process to define a region of the calibration grid you want to learn. The software ignores dot centers outside this region when it estimates the transformation. Creating a user-defined ROI is an effective way to increase correction speeds depending on the other calibration options selected. Set the user-defined ROI using the **ROI** parameter of either `imaqLearnCalibrationGrid()` or `imaqLearnCalibrationPoints()`.

## Choosing a Learning Algorithm

Select a method in which to learn the calibration information: perspective projection or nonlinear. Figure 6-5 illustrates the types of errors your image can exhibit. Figure 6-5a shows an image of a calibration grid with no errors. Figure 6-5b shows an image of a calibration grid with perspective projection. Figure 6-5c shows an image of a calibration grid with nonlinear distortion.

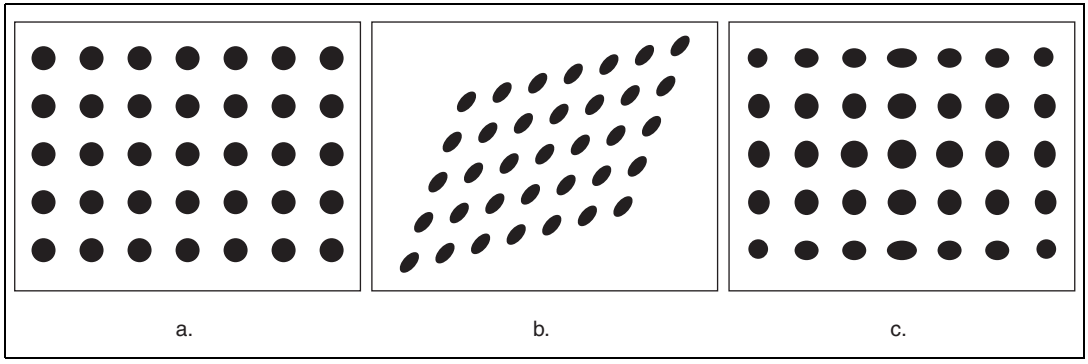
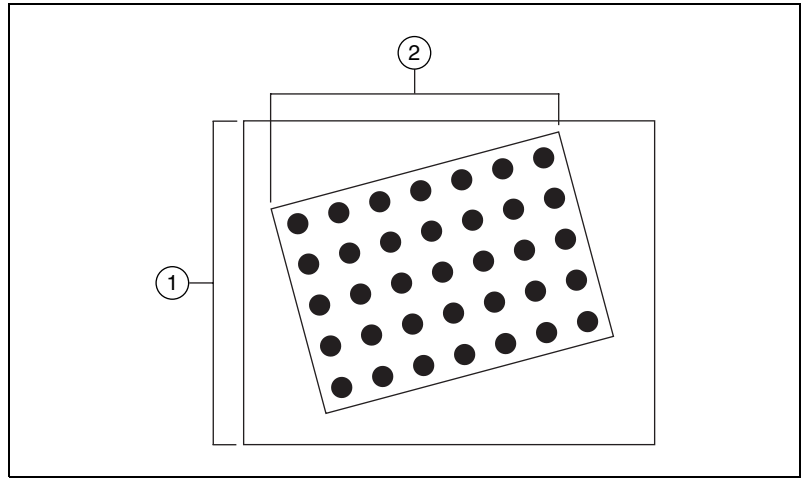



**Figure 6-5.** Types of Image Distortion

Choose the perspective projection algorithm when your system exhibits perspective errors only. A perspective projection calibration has an accurate transformation even in areas not covered by the calibration grid, as shown in Figure 6-6. Set the `mode` element of the **options** parameter to

IMAQ_PERSPECTIVE to choose the perspective calibration algorithm. Learning and applying perspective projection is less computationally intensive than the nonlinear method. However, perspective projection cannot handle nonlinear distortions.

If your imaging setup exhibits nonlinear distortion, use the nonlinear method. The nonlinear method guarantees accurate results only in the area that the calibration grid covers, as shown in Figure 6-6. If your system exhibits both perspective and nonlinear distortion, use the nonlinear method to correct for both. Set the mode element of the **options** parameter to IMAQ_NONLINEAR to choose the nonlinear calibration algorithm.




| | | | |
|---|---|---|---|
| 1 | Calibration ROI Using the Perspective Algorithm | 2 | Calibration ROI Using the Nonlinear Algorithm |

**Figure 6-6.** Calibration ROIs

## Using the Learning Score

The learning process returns a score that reflects how well the software learned the input image. A high learning score indicates that you chose the the appropriate learning algorithm, that the grid image complies with the guideline, and that your vision system setup is adequate.

If the learning process returns a low score, try the following:

1. Make sure your grid complies with the guidelines listed in the *Defining a Calibration Template* section.
2. Check the lighting conditions. If you have too much or too little lighting, the software may estimate the center of the dots incorrectly.

Also, adjust the **range** parameter to distinguish the dots from the background.

3. Select another learning algorithm. When nonlinear lens distortion is present, using perspective projection sometimes results in a low learning score.

**Note** A high score does *not* reflect the accuracy of your system.

## Learning the Error Map

An error map helps you gauge the quality of your complete system. The error map returns an estimated error range to expect when a pixel coordinate is transformed into a real-world coordinate. The transformation accuracy may be higher than the value the error range indicates. Set the `learnMap` element of the **options** parameter to `TRUE` to learn the error map.

## Learning the Correction Table

If the speed of image correction is a critical factor for your application, use a correction table. The correction table is a lookup table stored in memory that contains the real-world location information of all the pixels in the image. The extra memory requirements for this option are based on the size of the image. Use this option when you want to correct several images at a time in your vision application. Set the `learnTable` element of the **options** parameter to `TRUE` to learn the correction table.

## Setting the Scaling Method

Use the `method` element of the **options** parameter to choose the appearance of the corrected image. Select either `IMAQ_SCALE_TO_FIT` or `IMAQ_SCALE_TO_PRESERVE_AREA`. For more information about the scaling methods, see Chapter 3, *System Setup and Calibration*, in the *IMAQ Vision Concepts Manual*.

## Calibration Invalidation

Any image processing operation that changes the image size or orientation voids the calibration information in a calibrated image. Examples of functions that void calibration information include `imaqResample()`, `imaqScale()`, `imaqArrayToImage()`, and `imaqUnwrap()`.

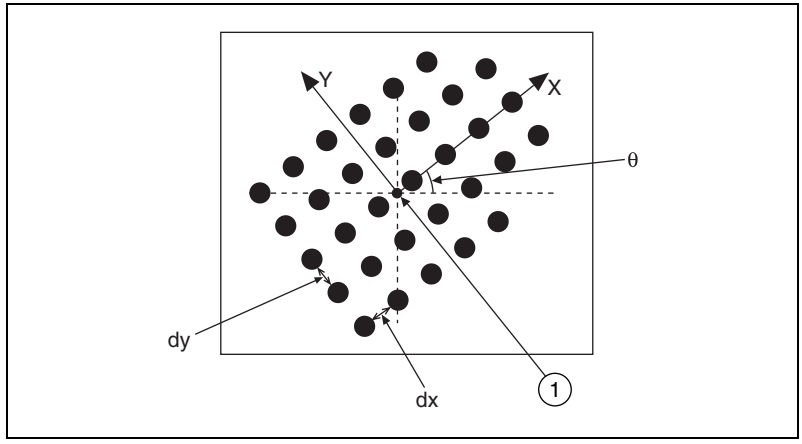
# Simple Calibration

When the axis of your camera is perpendicular to the image plane and lens distortion is negligible, use simple calibration. In simple calibration, a pixel coordinate is transformed to a real-world coordinate through scaling in the horizontal and vertical directions.

Use simple calibration to map pixel coordinates to real-world coordinates directly without a calibration grid. The software rotates and scales a pixel coordinate according to predefined coordinate reference and scaling factors. You can assign the calibration to an arbitrary image using `imaqSetSimpleCalibration()`.

To perform a simple calibration, set a coordinate reference (angle, center, and axis direction) and scaling factors on the defined axis, as shown in Figure 6-7. Express the angle between the x-axis and the horizontal axis of the image in degrees. Express the center as the position, in pixels, where you want the coordinate reference origin. Set the axis direction to direct or indirect. Simple calibration also offers a correction table option and a scaling mode option.

Use the **system** parameter to define the coordinate system. Use the **grid** parameter to specify the scaling factors. Use the **method** parameter to set the scaling method. Set the **learnTable** parameter to `TRUE` to learn the correction table.




1 Origin

**Figure 6-7.** Defining a Simple Calibration

# Save Calibration Information

After you learn the calibration information, you can save it so that you do not have to relearn the information for subsequent processing. Use `imaqWriteVisionFile()` to save the image of the grid and its associated calibration information to a file. To read the file containing the calibration information use `imaqReadVisionFile()`. For more information about attaching the calibration information you read from another image, see the *Attach Calibration Information* section.

# Attach Calibration Information

Now that you have calibrated your setup correctly, you can apply the calibration settings to images that you acquire. Use `imaqCopyCalibrationInfo()` to attach the calibration information of the current setup to each image you acquire. This function takes in a source image containing the calibration information and a destination image that you want to calibrate. The destination image is your inspection image with the calibration information attached to it.

Using the calibration information attached to the image, you can accurately convert pixel coordinates to real-world coordinates to make any of the analytic geometry measurements with `imaqTransformPixelToRealWorld()`. If your application requires that you make shape measurements, correct the image by removing distortion with `imaqCorrectImage()`.

**Note** Because calibration information is part of the image, it is propagated throughout the processing and analysis of the image. Functions that modify the image size (such as an image rotation function) void the calibration information. Use `imaqWriteVisionFile()` to save the image and all of the attached calibration information to a file.

# A

# Technical Support Resources

## Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`

## NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

## Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

## System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`

# Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

# Glossary

## Numbers/Symbols

| | |
|---|---|
| 1D | One-dimensional. |
| 2D | Two-dimensional. |
| 3D | Three-dimensional. |

## A

| | |
|---|---|
| AIPD | National Instruments proprietary image file format used for saving complex images and calibration information pertaining to step and spatial units (extension APD). |
| alignment | The process by which a machine vision application determines the location, orientation, and scale of a part being inspected. |
| alpha channel | Channel used to code extra information, such as gamma correction, about a color image. The alpha channel is stored as the first byte in the four-byte representation of an RGB pixel. |
| annulus | A region of interest that resembles a ring or partial ring. |
| area | (1) A rectangular portion of an acquisition window or frame that is controlled and defined by software. (2) The size of an object in pixels or user-defined units. |
| area threshold | Detects objects based on their size, which can fall within a user-specified range. |
| arithmetic operators | The image operations multiply, divide, add, subtract, and remainder. |
| array | Ordered, indexed set of data elements of the same type. |
| auto-median function | A function that uses dual combinations of opening and closing operations to smooth the boundaries of objects. |

# B

| | |
|---|---|
| b | Bit. One binary digit, either 0 or 1. |
| B | Byte. Eight related bits of data, an eight-bit binary number. Also denotes the amount of memory required to store one byte of data. |
| barycenter | The grayscale value representing the centroid of the range of an image's grayscale values in the image histogram. |
| binary image | An image in which the objects usually have a pixel intensity of 1 (or 255) and the background has a pixel intensity of 0. |
| binary morphology | Functions that perform morphological operations on a binary image. |
| binary threshold | Separation of an image into objects of interest (assigned a non-zero pixel value) and background (assigned pixel values of 0) based on the intensities of the image pixels. |
| bit depth | The number of bits ($n$) used to encode the value of a pixel. For a given $n$, a pixel can take $2^n$ different values. For example, if $n$ equals 8-bits, a pixel can take 256 different values ranging from 0 to 255. If $n$ equals 16 bits, a pixel can take 65,536 different values ranging from 0 to 65,535 or –32,768 to 32,767. |
| black reference level | The level that represents the darkest an image can get. *See also* white reference level. |
| blob | Binary large object. A connected region or grouping of pixels in an image in which all pixels have the same intensity level. Blobs are also referred to as objects or particles. |
| blob analysis | A series of processing operations and analysis functions that produce some information about the blobs in an image. |
| blurring | Reduces the amount of detail in an image. Blurring commonly occurs because the camera is out of focus. You can blur an image intentionally by applying a lowpass frequency filter. |
| BMP | Bitmap. Image file format commonly used for 8-bit and color images (extension BMP). |

| | |
|---|---|
| brightness | (1) A constant added to the red, green, and blue components of a color pixel during the color decoding process. (2) The perception by which white objects are distinguished from gray and light objects from dark objects. |
| buffer | Temporary storage for acquired data. |

## C

| | |
|---|---|
| caliper | (1) A function in IMAQ Vision Builder that calculates distances, angles, circular fits, and the center of mass based on positions given by edge detection, particle analysis, centroid, and search functions.<br>(2) A measurement function that finds edge pairs along a specified path in the image. This function performs an edge extraction and then finds edge pairs based on specified criteria such as the distance between the leading and trailing edges, edge contrasts, and so forth. |
| center of mass | The point on an object where all the mass of the object could be concentrated without changing the first moment of the object about any axis |
| centroid | (1) The average of the x-coordinates and y-coordinates of a binary image or a blob in the image. The centroid of a blob may lie outside the blob.<br>(2) The weighted average of the x-coordinates and y-coordinates in a grayscale image, where the weights are determined by the pixel values in the image. |
| character recognition | The ability of a machine to read human-readable text. |
| chroma | The color information in a video signal. |
| chromaticity | The combination of hue and saturation. The relationship between chromaticity and brightness characterizes a color. |
| circle function | Detects circular objects in a binary image. |
| closing | A dilation followed by an erosion. A closing fills small holes in objects and smooths the boundaries of objects. |
| clustering | A technique where the image is sorted into a discrete number of classes. Each class contains pixels that fall within a distinct range of grayscale values. The barycenter is determined for each class. This process is repeated until a value is obtained that represents the center of mass for each phase or class. |

| | |
|---|---|
| CLUT | Color lookup table. Table for converting the value of a pixel in an image into a red, green, and blue (RGB) intensity. |
| color images | Images containing color information, usually encoded in the RGB form. |
| color location | The technique that locates a color template in a color image based on only the color information. |
| color matching | The technique that compares the color information in an image or region of an image to the color information in another image or region of an image. |
| color pattern matching | The technique that locates a color template in a color image based on both the color information and grayscale pattern in the template. |
| color space | The mathematical representation for a color. For example, color can be described in terms of red, green, and blue; hue, saturation, and luminance; or hue, saturation, and intensity. |
| complex image | Stores information obtained from the FFT of an image. The complex numbers that compose the FFT plane are encoded in 64-bit floating-point values: 32bits for the real part and 32bits for the imaginary part. |
| connectivity | Defines which of the surrounding pixels of a given pixel constitute its neighborhood. |
| connectivity-4 | Only pixels adjacent in the horizontal and vertical directions are considered neighbors. |
| connectivity-8 | All adjacent pixels are considered neighbors. |
| contrast | (1) A constant multiplication factor applied to the luma and chroma components of a color pixel in the color decoding process.<br>(2) The difference between light and dark intensity values in an image. |
| convex function | Computes the convex regions of objects in a binary image. |
| convex hull | The smallest convex polygon that can encapsulate a particle. |
| convolution | *See* linear filter. |
| convolution kernel | 2D matrices (or templates) used to represent the filter in the filtering process. The contents of these kernels are a discrete two-dimensional representation of the impulse response of the filter that they represent. |

| | |
|---|---|
| cross correlation | A technique that compares the similarity of two images or parts of an image. You can use cross correlation to find the optimal position where similarity exists. |

## D

| | |
|---|---|
| Danielsson function | Similar to the distance functions, but with more accurate results. |
| dB | Decibel. The unit for expressing a logarithmic measure of the ratio of two signal levels: dB = 20log10 V1/V2, for signals in volts. |
| default setting | A default parameter value recorded in the driver. In many cases, the default input of a control is a certain value (often 0). |
| definition | The number of values a pixel can take on, which is the number of colors or shades that you can see in the image. |
| dendrite | Branches of the skeleton of an object. |
| densitometry | Intensity information about an image or regions of an image. Typical measurements include minimum, maximum, and mean intensity values as well as the standard deviation of the intensity values. |
| density function | For each gray level in a linear histogram, the function gives the number of pixels in the image that have the same gray level. |
| differentiation filter | Extracts the contours (edge detection) in gray level. |
| digital image | An image f (x, y) that has been converted into a discrete number of pixels. Both spatial coordinates and brightness are specified. |
| dilation | Increases the size of an object along its boundary and removes tiny holes in the object. |
| distance calibration | Determination of the physical dimensions of a pixel by defining the physical dimensions of a line in the image. |
| distance function | Assigns to each pixel in an object a gray-level value equal to its shortest Euclidean distance from the border of the object. |
| driver | Software that controls a specific hardware device, such as an IMAQ or DAQ device. |

# E

| | |
|---|---|
| edge | Defined by a sharp change (transition) in the pixel intensities in an image or along an array of pixels. |
| edge contrast | The difference between the average pixel intensity before and the average pixel intensity after the edge. |
| edge detection | Any of several techniques to identify the edges of objects in an image. |
| edge hysteresis | The difference in threshold level between a rising and a falling edge. |
| edge steepness | The number of pixels that corresponds to the slope or transition area of an edge. |
| energy center | The center of mass of a grayscale image. *See* center of mass. |
| entropy | A measure of the randomness in an image. An image with high entropy contains more pixel value variation than an image with low entropy. |
| equalize function | *See* histogram equalization. |
| erosion | Reduces the size of an object along its boundary and eliminates isolated points in the image. |
| Euclidean distance | The shortest distance between two points in a Cartesian system. |
| exponential and gamma corrections | Expand the high gray-level information in an image while suppressing low gray-level information. |
| exponential function | Decreases brightness and increases contrast in bright regions of an image, and decreases contrast in dark regions of an image. |

# F

| | |
|---|---|
| feature | A specific pattern of pixels in an image. |
| FFT | Fast Fourier Transform. A method used to compute the Fourier transform of an image to get frequency information. |
| fiducial | A reference pattern on a part that helps a machine vision application find the part's location and orientation in an image. |

| | |
|---|---|
| form | Window or area on the screen on which you place controls and indicators to create the user interface for your program. |
| Fourier spectrum | The magnitude information of the Fourier transform of an image. |
| Fourier transform | Transforms an image from the spatial domain to the frequency domain. |
| frequency filters | Counterparts of spatial filters in the frequency domain. For images, frequency information is in the form of spatial frequency. |
| function | A set of software instructions executed by a single line of code that may have input and/or output parameters and returns a value when executed. |

## G

| | |
|---|---|
| gamma | The nonlinear change in the difference between the video signal's brightness level and the voltage level needed to produce that brightness. |
| gauging | Measurement of an object or distances between objects. |
| Gaussian filter | A filter similar to the smoothing filter, but using a Gaussian kernel in the filter operation. The blurring in a Gaussian filter is more gentle than a smoothing filter. |
| gradient convolution filter | *See* gradient filter. |
| gradient filter | Extracts the contours (edge detection) in gray-level values. Gradient filters include the Prewitt and Sobel filters. |
| gray level | The brightness of a pixel in an image. |
| gray-level dilation | Increases the brightness of pixels in an image that are surrounded by other pixels with a higher intensity. |
| gray-level erosion | Reduces the brightness of pixels in an image that are surrounded by other pixels with a lower intensity. |
| grayscale image | An image with monochrome information. |
| grayscale morphology | Functions that perform morphological operations on a gray-level image. |

# H

| | |
|---|---|
| highpass attenuation | Applies a linear attenuation to the frequencies in an image, with no attenuation at the highest frequency and full attenuation at the lowest frequency. |
| highpass FFT filter | Removes or attenuates low frequencies present in the FFT domain of an image. |
| highpass filter | Emphasizes the intensity variations in an image, detects edges (or object boundaries), and enhances fine details in an image. |
| highpass frequency filter | Attenuates or removes (truncates) low frequencies present in the frequency domain of the image. A highpass frequency filter suppresses information related to slow variations of light intensities in the spatial image. |
| highpass truncation | Removes all frequency information below a certain frequency. |
| histogram | Indicates the quantitative distribution of the pixels of an image per gray-level value. |
| histogram equalization | Transforms the gray-level values of the pixels of an image to occupy the entire range (0 to 255 in an 8-bit image) of the histogram, increasing the contrast of the image. |
| histogram inversion | Finds the inverse of an image. The histogram of a reversed image is equal to the original histogram flipped horizontally around the center of the histogram. |
| hit-miss function | Locates objects in the image similar to the pattern defined in the structuring element. |
| hole filling function | Fills all holes in objects that are present in a binary image. |
| HSI | Color encoding scheme in Hue, Saturation, and Intensity. |
| HSL | Color encoding scheme using Hue, Saturation, and Luma information where each image in the pixel is encoded using 32 bits: 8 bits for hue, 8 bits for saturation, 8 bits for luma, and 8 bits for the alpha channel. |
| HSV | Color encoding scheme in Hue, Saturation, and Value. |
| hue | Represents the dominant color of a pixel. The hue function is a continuous function that covers all the possible colors generated using the R, G, and B primaries. *See also* RGB. |

| | |
|---|---|
| hue offset angle | The value added to all hue values so that the discontinuity occurs outside the values of interest during analysis. |

## I

| | |
|---|---|
| image | A two-dimensional light intensity function f (x, y) where x and y denote spatial coordinates and the value f at any point (x, y) is proportional to the brightness at that point. |
| image border | A user-defined region of pixels surrounding an image. Functions that process pixels based on the value of the pixel neighbors require image borders. |
| Image Browser | An image that contains thumbnails of images to analyze or process in a vision application. |
| image buffer | Memory location used to store images. |
| image definition | *See* pixel depth. |
| image enhancement | The process of improving the quality of an image that you acquire from a sensor in terms of signal-to-noise ratio, image contrast, edge definition, and so on. |
| image file | A file containing pixel data and additional information about the image. |
| image format | Defines how an image is stored in a file. Usually composed of a header followed by the pixel data. |
| image mask | A binary image that isolates parts of a source image for further processing. A pixel in the source image is processed if its corresponding mask pixel has a non-zero value. A source pixel whose corresponding mask pixel has a value of 0 is left unchanged. |
| image palette | The gradation of colors used to display an image on screen, usually defined by a color lookup table. |
| image processing | Encompasses various processes and analysis functions that you can apply to an image. |
| image source | Original input image. |
| image understanding | A technique that interprets the content of the image at a symbolic level rather than a pixel level. |

| | |
|---|---|
| image visualization | The presentation (display) of an image (image data) to the user. |
| imaging | Any process of acquiring and displaying images and analyzing image data. |
| IMAQ | Image Acquisition. |
| inner gradient | Finds the inner boundary of objects. |
| inspection | The process by which parts are tested for simple defects such as missing parts or cracks on part surfaces. |
| inspection function | Analyzes groups of pixels within an image and returns information about the size, shape, position, and pixel connectivity. Typical applications include quality of parts, analyzing defects, locating objects, and sorting objects. |
| instrument driver | A set of high-level software functions, such as NI-IMAQ, that control specific plug-in computer boards. Instrument drivers are available in several forms, ranging from a function callable from a programming language to a virtual instrument (VI) in LabVIEW. |
| intensity | (1) The sum of the Red, Green, and Blue primary colors divided by three, (*Red* + *Green* + *Blue*)/3 in a color image. (2) The gray-level value of a pixel in a grayscale image. |
| intensity calibration | Assigning user-defined quantities, such as optical densities or concentrations, to the gray-level values in an image. |
| intensity profile | The gray-level distribution of the pixels in an ROI of an image. |
| intensity range | Defines the range of gray-level values in an object of an image. |
| intensity threshold | Characterizes an object based on the range of gray-level values in the object. If the intensity range of the object falls within the user-specified range, it is considered an object. Otherwise it is considered part of the background. |
| interpolation | The technique used to find values in between known values when resampling an image or array of pixels. |
| IRE | A relative unit of measure (named for the Institute of Radio Engineers). 0 IRE corresponds to the blanking level of a video signal, 100 IRE to the white level. Note that for CCIR/PAL video, the black level is equal to the blanking level or 0 IRE, while for RS-170/NTSC video, the black level is at 7.5 IRE. |

## J

| | |
|---|---|
| JPEG | Joint Photographic Experts Group. Image file format for storing 8-bit and color images with lossy compression (extension JPG). |

## K

| | |
|---|---|
| kernel | Structure that represents a pixel and its relationship to its neighbors. The relationship is specified by weighted coefficients of each neighbor. |

## L

| | |
|---|---|
| labeling | The process by which each object in a binary image is assigned a unique value. This process is useful for identifying the number of objects in the image and giving each object a unique identity. |
| LabVIEW | Laboratory Virtual Instrument Engineering Workbench. Program development environment application based on the programming language G used commonly for test and measurement applications. |
| Laplacian filter | Extracts the contours of objects in the image by highlighting the variation of light intensity surrounding a pixel. |
| line gauge | Measures the distance between selected edges with high-precision subpixel accuracy along a line in an image. For example, this function can be used to measure distances between points and edges. This function also can step and repeat its measurements across the image. |
| line profile | Represents the gray-level distribution along a line of pixels in an image. |
| linear filter | A special algorithm that calculates the value of a pixel based on its own pixel value as well as the pixel values of its neighbors. The sum of this calculation is divided by the sum of the elements in the matrix to obtain a new pixel value. |
| logarithmic and inverse gamma corrections | Expand low gray-level information in an image while compressing information from the high gray-level ranges. |
| logarithmic function | Increases the brightness and contrast in dark regions of an image and decreases the contrast in bright regions of the image. |

| | |
|---|---|
| logic operators | The image operations AND, NAND, OR, XOR, NOR, XNOR, difference, mask, mean, max, and min. |
| lossless compression | Compression in which the decompressed image is identical to the original image. |
| lossy compression | Compression in which the decompressed image is visually similar but not identical to the original image. |
| lowpass attenuation | Applies a linear attenuation to the frequencies in an image, with no attenuation at the lowest frequency and full attenuation at the highest frequency. |
| lowpass FFT filter | Removes or attenuates high frequencies present in the FFT domain of an image. |
| lowpass filter | Attenuates intensity variations in an image. You can use these filters to smooth an image by eliminating fine details and blurring edges. |
| lowpass frequency filter | Attenuates high frequencies present in the frequency domain of the image. A lowpass frequency filter suppresses information related to fast variations of light intensities in the spatial image. |
| lowpass truncation | Removes all frequency information above a certain frequency. |
| LSB | Least significant bit. |
| L-skeleton function | Uses an L-shaped structuring element in the skeleton function. |
| luma | The brightness information in the video picture. The luma signal amplitude varies in proportion to the brightness of the video signal and corresponds exactly to the monochrome picture. |
| luminance | *See* luma. |
| LUT | Lookup table. Table containing values used to transform the gray-level values of an image. For each gray-level value in the image, the corresponding new value is obtained from the lookup table. |

## M

| | |
|---|---|
| machine vision | An automated application that performs a set of visual inspection tasks. |
| mask FFT filter | Removes frequencies contained in a mask (range) specified by the user. |

| | |
|---|---|
| match score | A number ranging from 0 to 1000 that indicates how closely an acquired image matches the template image. A match score of 1000 indicates a perfect match. A match score of 0 indicates no match. |
| median filter | A lowpass filter that assigns to each pixel the median value of its neighbors. This filter effectively removes isolated pixels without blurring the contours of objects. |
| memory buffer | *See* buffer. |
| morphological transformations | Extract and alter the structure of objects in an image. You can use these transformations for expanding (dilating) or reducing (eroding) objects, filling holes, closing inclusions, or smoothing borders. They are used primarily to delineate objects and prepare them for quantitative inspection analysis. |
| MSB | Most significant bit. |
| M-skeleton function | Uses an M-shaped structuring element in the skeleton function. |

## N

| | |
|---|---|
| neighbor | A pixel whose value affects the value of a nearby pixel when an image is processed. The neighbors of a pixel are usually defined by a kernel or a structuring element. |
| neighborhood operations | Operations on a point in an image that take into consideration the values of the pixels neighboring that point. |
| NI-IMAQ | Driver software for National Instruments IMAQ hardware. |
| nonlinear filter | Replaces each pixel value with a nonlinear function of its surrounding pixels. |
| nonlinear gradient filter | A highpass edge-extraction filter that favors vertical edges. |
| nonlinear Prewitt filter | A highpass, edge-extraction filter based on two-dimensional gradient information. |
| nonlinear Sobel filter | A highpass, edge-extraction filter based on two-dimensional gradient information. The filter has a smoothing effect that reduces noise enhancements caused by gradient operators. |

| | |
|---|---|
| Nth order filter | Filters an image using a nonlinear filter. This filter orders (or classifies) the pixel values surrounding the pixel being processed. The pixel being processed is set to the Nth pixel value, where N is the order of the filter. |
| number of planes (in an image) | The number of arrays of pixels that compose the image. A gray-level or pseudo-color image is composed of one plane, while an RGB image is composed of three planes (one for the red component, one for the blue, and one for the green). |

## O

| | |
|---|---|
| object | A connected region or grouping of pixels in an image in which all pixels have the same intensity level. Objects are also referred to as blobs or particles. |
| offset | The coordinate position in an image where you want to place the origin of another image. Setting an offset is useful when performing mask operations. |
| opening | An erosion followed by a dilation. An opening removes small objects and smooths boundaries of objects in the image. |
| operators | Allow masking, combination, and comparison of images. You can use arithmetic and logic operators in IMAQ Vision. |
| optical character verification | A machine vision application that inspects the quality of printed characters. |
| optical representation | Contains the low-frequency information at the center and the high-frequency information at the corners of an FFT-transformed image. |
| outer gradient | Finds the outer boundary of objects. |
| overlay | ROIs, text, and bitmaps that you can place on top of a displayed image to annotate it without modifying it. |

## P

| | |
|---|---|
| palette | The gradation of colors used to display an image on screen, usually defined by a color lookup table. |

| | |
|---|---|
| particle | A connected region or grouping of pixels in an image in which all pixels have the same intensity level. Particles are also referred to as blobs or objects. |
| pattern matching | The technique used to quickly locate a grayscale template within a grayscale image |
| picture aspect ratio | The ratio of the active pixel region to the active line region. For standard video signals like RS-170 or CCIR, the full-size picture aspect ratio normally is 4/3 (1.33). |
| picture element | An element of a digital image. Also called pixel. |
| pixel | Picture element. The smallest division that makes up the video scan line. For display on a computer monitor, a pixel's optimum dimension is square (aspect ratio of 1:1, or the width equal to the height). |
| pixel aspect ratio | The ratio between the physical horizontal size and the vertical size of the region covered by the pixel. An acquired pixel should optimally be square, thus the optimal value is 1.0, but typically it falls between 0.95 and 1.05, depending on camera quality. |
| pixel calibration | Directly calibrating the physical dimensions of a pixel in an image. |
| pixel depth | The number of bits used to represent the gray level of a pixel. |
| PNG | Portable Network Graphic. Image file format for storing 8-bit, 16-bit, and color images with lossless compression (extension PNG). |
| power 1/Y function | Similar to a logarithmic function but with a weaker effect. |
| power Y function | *See* exponential function. |
| Prewitt filter | Extracts the contours (edge detection) in gray-level values using a 3 x 3 filter kernel. *See* gradient filter. |
| probability function | Defines the probability that a pixel in an image has a certain gray-level value. |
| proper-closing | A finite combination of successive closing and opening operations that you can use to fill small holes and smooth the boundaries of objects. |
| proper-opening | A finite combination of successive opening and closing operations that you can use to remove small particles and smooth the boundaries of objects. |

pyramidal matching — A technique used to increase the speed of a pattern matching algorithm by matching subsampled versions of the image and the reference pattern.

## Q

quantitative analysis — Obtaining various measurements of objects in an image.

## R

real time — A property of an event or system in which data is processed as it is acquired instead of being accumulated and processed at a later time.

relative accuracy — A measure in LSB of the accuracy of an ADC; it includes all nonlinearity and quantization errors but does not include offset and gain errors of the circuitry feeding the ADC.

remove border objects function — Removes blobs in a binary image that touch the image border.

resolution — The number of rows and columns of pixels. An image composed of m rows and n columns has a resolution of $m \times n$.

reverse function — Inverts the pixel values in an image.

RGB — Color encoding scheme using red, green, and blue (RGB) color information where each pixel in the color image is encoded using 32 bits: 8 bits for red, 8 bits for green, 8 bits for blue, and 8 bits for the alpha value (unused).

Roberts filter — Extracts the contours (edge detection) in gray level, favoring diagonal edges.

ROI — Region of interest. (1) An area of the image that is graphically selected from a window displaying the image. This area can be used to focus further processing. (2) A hardware-programmable rectangular portion of the acquisition window.

ROI tools — Collection of tools that enable you to select a region of interest from an image. These tools let you select points, lines, annuli, polygons, rectangles, rotated rectangles, ovals, and freehand open and closed contours.

rotational shift — The amount by which one image is rotated with respect to a reference image. This rotation is computed with respect to the center of the image.

| | |
|---|---|
| rotation-invariant matching | A pattern matching technique in which the reference pattern can be located at any orientation in the test image as well as rotated at any degree. |

## S

| | |
|---|---|
| saturation | The amount of white added to a pure color. Saturation relates to the richness of a color. A saturation of zero corresponds to a pure color with no white added. Pink is a red with low saturation. |
| scale-invariant matching | A pattern matching technique in which the reference pattern can be any size in the test image. *See* shape matching. |
| segmentation function | Fully partitions a labeled binary image into non-overlapping segments, with each segment containing a unique object. |
| separation function | Separates objects that touch each other by narrow isthmuses. |
| shape matching | Finds objects in an image whose shape matches the shape of the object specified by a shape template. The matching process is invariant to rotation and can be set to be invariant to the scale of the objects. |
| shift-invariant matching | A pattern matching technique in which the reference pattern can be located anywhere in the test image but cannot be rotated or scaled. |
| Sigma filter | A highpass filter that outlines edges. |
| skeleton function | Applies a succession of thinning operations to an object until its width becomes one pixel. |
| skiz function | Obtains lines in an image that separate each object from the others and are equidistant from the objects that they separate. |
| smoothing filter | Blurs an image by attenuating variations of light intensity in the neighborhood of a pixel. |
| Sobel filter | Extracts the contours (edge detection) in gray-level values using a $3 \times 3$ filter kernel. *See* gradient filter. |
| spatial calibration | Assigning physical dimensions to the area of a pixel in an image. |
| spatial filters | Alter the intensity of a pixel with respect to variations in intensities of its neighboring pixels. You can use these filters for edge detection, image enhancement, noise reduction, smoothing, and so forth. |

| | |
|---|---|
| spatial resolution | The number of pixels in an image, in terms of the number of rows and columns in the image. |
| square function | *See* exponential function. |
| square root function | *See* logarithmic function. |
| standard representation | Contains the low-frequency information at the corners and high-frequency information at the center of an FFT-transformed image. |
| structuring element | A binary mask used in most morphological operations. A structuring element is used to determine which neighboring pixels contribute in the operation. |
| sub-pixel analysis | Finds the location of the edge coordinates in terms of fractions of a pixel. |

## T

| | |
|---|---|
| template | Color, shape, or pattern that you are trying to match in an image using the color matching, shape matching, or pattern matching functions. A template can be a region selected from an image or it can be an entire image. |
| thickening | Alters the shape of objects by adding parts to the object that match the pattern specified in the structuring element. |
| thinning | Alters the shape of objects by eliminating parts of the object that match the pattern specified in the structuring element. |
| threshold | Separates objects from the background by assigning all pixels with intensities within a specified range to the object and the rest of the pixels to the background. In the resulting binary image, objects are represented with a pixel intensity of 255 and the background is set to 0. |
| threshold interval | Two parameters, the lower threshold gray-level value and the upper threshold gray-level value. |
| TIFF | Tagged Image File Format. Image format commonly used for encoding 8-bit, 16-bit, and color images (extension TIF). |
| Tools palette | Collection of tools that enable you to select regions of interest, zoom in and out, and change the image palette. |

## V

| | |
|---|---|
| value | The grayscale intensity of a color pixel computed as the average of the maximum and minimum red, green, and blue values of that pixel. |

## W

| | |
|---|---|
| watershed | A technique used to segment an image into multiple regions. |
| web inspection | The process of detecting defects in a continuous sheet of materials at production speeds. Example materials include plastic film, cloth, paper and pulp products, metal, and glass. |
| white reference level | The level that defines what is white for a particular video system. *See also* black reference level. |

# Index

## A



## B



## C

## D



## E



## F

## G



## H



## I

## L



## M

## N



## O



## P



## R

## S



## T



## V



## W



## Z